HashEx
Blockchain Security

HurricaneSwap

smart contracts
final audit report

December 2021

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

2. Overview

HashEx was commissioned by the HurricaneSwap team to perform an audit of their smart contracts. The audit was conducted between 12.11.2021 and 23.11.2021.

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 was provided via private GitHub repositories. Project documentation is available at hurricaneswap.gitbook.io. The recheck was performed on the HurricaneSwap GitHub repositories.

2.1 Summary

Project name	HurricaneSwap
URL	https://hurricaneswap.com
Platform	Binance Smart Chain, Avalanche Network
Language	Solidity

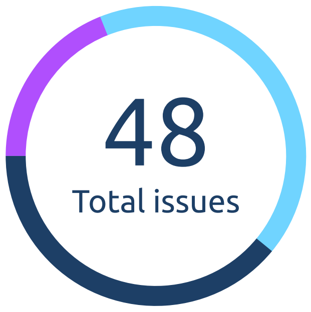
2.2 Contracts

Name	Address
HOGWalletAVAX	https://github.com/HurricaneSwap/v2-arbitrage-contract/blob/68afe585abfda80bf0823756f8598f7fc0d7a680/avax/HOGWalletAVAX.sol
HOGWalletBSC	https://github.com/HurricaneSwap/v2-arbitrage-contract/blob/68afe585abfda80bf0823756f8598f7fc0d7a680/bsc/HOGWalletBSC.sol
PancakeArbitrageHelper	https://github.com/HurricaneSwap/v2-arbitrage-contract/blob/68afe585abfda80bf0823756f8598f7fc0d7a680/bsc/PancakeArbitrageHelper.sol
LPMigration	0xab33EB3a233776fE389301358b8432a7a35F95C9
StakingRewards	https://github.com/HurricaneSwap/hct-contract/blob/e019f0ebebdfe2c2c8e1e5464affb4fe10484c04/reward/StakingRewardERC20.sol
HcSwapAvaxRouter	0xB6559dD3cBec057078fBd9477e676a24cCEA0609
LPBridgeHub	0x7009b3619d5ee60d0665BA27Cf85eDF95fd8Ad01
BridgeToken	0x4691aF9D439df39473B1698076793a975Ebc497f
HurricaneArbitrageHelper	https://github.com/HurricaneSwap/v2-arbitrage-contract/blob/68afe585abfda80bf0823756f8598f7fc0d7a680/avax/HurricaneArbitrageHelper.sol
LPBridgeRemote	0x4691aF9D439df39473B1698076793a975Ebc497f

HcSwapAvaxPair	https://github.com/HurricaneSwap/v2-contract/blob/60f4059cea65616ed4e3fc30bb9c1120d9c55e74/avax/HcSwapAvaxPair.sol
HcSwapV2Router02	0x38eF8ECfEe7A698bb7A8342e4b38B4513f9ad889
HcSwapBSCPair	https://github.com/HurricaneSwap/v2-contract/blob/60f4059cea65616ed4e3fc30bb9c1120d9c55e74/bsc/HcSwapBSCPair.sol
HcToken	https://github.com/HurricaneSwap/v2-contract/blob/1075e48edeb31a52e964ca1641cb83dba1e441ca/avax/HcToken.sol
PancakeLibrary	https://github.com/HurricaneSwap/v2-arbitrage-contract/blob/68afe585abfda80bf0823756f8598f7fc0d7a680/bsc/library/PancakeLibrary.sol
HcSwapFactory	0x75B09f7bA3a2F24501a3c707f3ba7C7E342D2558
HcSwapERC20	https://github.com/HurricaneSwap/v2-contract/blob/60f4059cea65616ed4e3fc30bb9c1120d9c55e74/bsc/HcSwapERC20.sol
Timelock	https://github.com/HurricaneSwap/v2-arbitrage-contract/blob/68afe585abfda80bf0823756f8598f7fc0d7a680/Timelock.sol
HcSwapAvaxFactory	0x7009b3619d5ee60d0665BA27Cf85eDF95fd8Ad01
HcTokenFactory	0x806aA90a161f18106Bd45033cE3E198E2Af28c35
LPQueue	https://github.com/HurricaneSwap/v2-contract/blob/1075e48edeb31a52e964ca1641cb83dba1e441ca/public/libraries/LPQueue.sol

HcSwapAvaxERC20	https://github.com/HurricaneSwap/v2-contract/blob/60f4059cea65616ed4e3fc30bb9c1120d9c55e74/avax/HcSwapAvaxERC20.sol
HurricaneLibrary	https://github.com/HurricaneSwap/v2-arbitrage-contract/blob/68afe585abfda80bf0823756f8598f7fc0d7a680/avax/library/HurricaneLibrary.sol
RewardsDistributionRecipient	https://github.com/HurricaneSwap/hct-contract/blob/e019f0ebebdfe2c2c8e1e5464affb4fe10484c04/reward/RewardsDistributionRecipient.sol
HcSwapHelper	0x5098328EAD41F41614468Aa5e6dd57e67aC4304D

3. Found issues



■	Medium	9 (19%)
■	Low	20 (42%)
■	Informational	19 (39%)

HOGWalletAVAX

ID	Title	Severity	Status
01	Lack of events in set functions	■ Low	Resolved
02	Gas optimization	■ Informational	Acknowledged
03	Unnecessary equality check	■ Informational	Resolved

HOGWalletBSC

ID	Title	Severity	Status
01	Lock logic violation	■ Medium	Resolved
02	Lack of events in set functions	■ Low	Resolved
03	Short minimum withdrawal delay	■ Low	Resolved

04	Gas optimizations	■ Informational	Partially fixed
05	Unnecessary equality check	■ Informational	Resolved

PancakeArbitrageHelper

ID	Title	Severity	Status
01	Swap's zero slippage	■ Medium	Partially fixed
02	Lack of events in set functions	■ Low	Acknowledged
03	Unnecessary equality check	■ Informational	Resolved
04	Extra calculation	■ Informational	Resolved
05	Gas optimizations	■ Informational	Partially fixed

LPMigration

ID	Title	Severity	Status
01	Slippage problem	■ Low	Acknowledged
02	Gas optimization	■ Low	Acknowledged

StakingRewards

ID	Title	Severity	Status
01	Transfer problem	■ Medium	Acknowledged
02	Reward amount	■ Low	Acknowledged

03	Gas optimizations	Low	Acknowledged
04	Withdraw logic	Informational	Acknowledged
05	Token support	Informational	Acknowledged

HcSwapAvaxRouter

ID	Title	Severity	Status
01	Owner overpower	Medium	Acknowledged
02	Token support	Informational	Acknowledged
03	Not enough events	Informational	Acknowledged

LPBridgeHub

ID	Title	Severity	Status
01	Floating pragma	Low	Acknowledged
02	Structure duplication	Low	Acknowledged
03	Lack of events in setter functions	Low	Acknowledged
04	DoS with block gas limit	Informational	Acknowledged

BridgeToken

ID	Title	Severity	Status
01	Floating pragma	■ Low	Acknowledged

HurricaneArbitrageHelper

ID	Title	Severity	Status
01	Swap's zero slippage	■ Medium	Acknowledged
02	Unnecessary equality check	■ Low	Resolved
03	Lack of events in set functions	■ Low	Acknowledged
04	Extra calculation	■ Informational	Resolved
05	Gas optimizations	■ Informational	Partially fixed
06	Arbitrage can be unrefunded	■ Informational	Acknowledged

LPBridgeRemote

ID	Title	Severity	Status
01	Initialized token mapping rewrite	■ Medium	Acknowledged
02	Gas optimization	■ Low	Acknowledged
03	Floating pragma	■ Low	Acknowledged
04	Lack of events	■ Low	Acknowledged

HcSwapAvaxPair

ID	Title	Severity	Status
01	Not enough events	■ Informational	Acknowledged
02	View function fail	■ Informational	Acknowledged

HcSwapV2Router02

ID	Title	Severity	Status
01	Owner overpower	■ Medium	Acknowledged
02	Not enough events	■ Low	Acknowledged
03	transfer() function	■ Low	Acknowledged
04	Token support	■ Informational	Acknowledged

HcSwapBSCPair

ID	Title	Severity	Status
01	Owner overpower	■ Medium	Acknowledged

HcToken

ID	Title	Severity	Status
01	Owner overpower	■ Medium	Acknowledged

02	Gas optimization	■ Low	Acknowledged

PancakeLibrary

ID	Title	Severity	Status
01	Fees are unconcerned in amounts calculations	■ Informational	Resolved

4. Contracts

4.1 HOGWalletAVAX

4.1.1 Overview

Part of BSC-AVAX bridge scheme.

4.1.2 Issues

01. Lack of events in set functions

- Low ✓ Resolved

When an arbitrager's (L:42 - L:46) or a tracer's (L:50 - L:54) status is set, proper events are not emitted.

02. Gas optimization

- Informational ! Acknowledged

CrossNonce variable is read multiple times in the crossChainToBSC() function.

03. Unnecessary equality check

- Informational ✓ Resolved

No need to check the returned boolean on equality to true/false (L:30, L:35).

4.2 HOGWalletBSC

4.2.1 Overview

Part of BSC-AVAX bridge scheme.

4.2.2 Issues

01. Lock logic violation

▪ Medium ✓ Resolved

MINIMUM_DELAY is subtracted from lockTs (L:104), while it should be added. This inappropriate arithmetical operation breaks the constraint on instant transaction withdrawal.

02. Lack of events in set functions

▪ Low ✓ Resolved

When an arbitrager's (L:42 - L:46) or a tracer's (L:50 - L:54) status is set or hurricaneRouter address is changed (L:63), proper events are not emitted.

03. Short minimum withdrawal delay

▪ Low ✓ Resolved

The minimum withdrawal delay of 12 hours is too short, we recommend using the standard one: 48 hours from Compound Timelock.

04. Gas optimizations

- Informational ◎ Partially fixed

a. The CrossNonce variable is read multiple times in the crossChainToAVAX() function.

b. The hurricaneRouter variable is read multiple times in the onCrossSync() function.

05. Unnecessary equality check

- Informational ✓ Resolved

No need to check the returned boolean on equality to true/false (L:35, L:40).

4.3 PancakeArbitrageHelper

4.3.1 Overview

BSC part of the arbitrage mechanism.

4.3.2 Issues

01. Swap's zero slippage

- Medium ◎ Partially fixed

Setting slippage to 0 (L:113) makes swaps on DEX vulnerable to a sandwich attack.

Recommendation

The slippage should be obtained via an external call to oracle or calculated on the backend.

Update

In the 7c81dfce commit HurricaneSwap team changed the doSwap() function signature. Although previous constant slippage is substituted with the minAmountOut parameter, it equals zero when doSwap() is called in _doArbitrage().

02. Lack of events in set functions

■ Low ⓘ Acknowledged

No events in setArbitrager(), setRouter(), setFactory(), setHOGWallet(), setWAVAX() functions (L:48-75).

03. Unnecessary equality check

■ Informational ✓ Resolved

No need to check the returned boolean on equality to true/false (L:26).

04. Extra calculation

- Informational ✓ Resolved

Calculation of the swap deadline (L:112) is unnecessary. Swap transactions will be in the same block, consequently, they will have the same timestamp as the arbitrage transaction.

05. Gas optimizations

- Informational ◌ Partially fixed

a. Unused variable WAVAX_Deprecated.

b. _router, wAVAX, _HOGWallet variables are read multiple times in the doSwap() function.

c. The wAVAX variable is read multiple times in the doArbitrage() function.

d. The _factory variable is read multiple times in the getReserves() function.

e. Excessive computations L:225, simple multiplication inequality may be used as it is in getArbitrageAmount().

4.4 LPMigration

4.4.1 Overview

The contract that makes migration of liquidity from one Uniswap-like DEX to another.

4.4.2 Issues

01. Slippage problem

- Low ⓘ Acknowledged

Because the contract uses one slippage for both operations (remove old LP tokens and add new LP tokens), it should be small enough to perform both of these operations. Slippage becomes smaller and smaller in case there already was a new LP pair in which the rate between tokens is different than the rate in the old pair (the one from which the migration originates) and the contract migrates to the existing LP pair. And when slippage decreases, more possibilities for performing sandwich attacks on the migration transaction become available.

02. Gas optimization

- Low ⓘ Acknowledged

Variables oldRouter and newRouter can be marked as immutable. By doing this, gas consumption can be reduced because the reading of this variable will be gas-free.

In the function migration() there are multiple reads of variables oldRouter and newRouter . Each reading from global states consumes gas and redundant reads of global variables increase the gas cost of transactions.

4.5 StakingRewards

4.5.1 Overview

Staking contract based on Synthetix.io staking model with minor modifications. Mints and burns ERC20 tokens with stake() and unstake().

4.5.2 Issues

01. Transfer problem

■ Medium ⓘ Acknowledged

Variable _stakeBalances should be removed or token transfers should be disabled. Because of this variable, only users who have staked tokens can unstake them. If a user accidentally transfers their tokens, then the stake for these tokens will be lost (it can be restored by receiving these tokens back).

02. Reward amount

■ Low ⓘ Acknowledged

The contract doesn't have guarantees that it holds enough reward tokens. Some users can lose some rewards if the contract doesn't have enough rewards. This is correct because:

1. Admin has access to all reward tokens in the contract and can withdraw any amount of reward tokens
2. In the function notifyRewardAmount() pending rewards are not taken into account, and because of that checking, that contract holds enough reward tokens with error. It requires less tokens than it actually needs.

03. Gas optimizations

- Low ⓘ Acknowledged

Variables rewardsToken and stakingToken can be marked as immutable. By doing this, gas consumption can be reduced because the reading of this variable will be gas-free.

Variables periodFinish and rewardRate are zero-initialized. Every variable, that doesn't have initialization from the beginning will be zero-initialized. Double zero-initializing of a global variable is a gas waste.

In the function lastTimeRewardApplicable() there is a double read of the variable periodFinish. Each reading from global states consumes gas and redundant reads of global variables increase the gas cost of transactions.

In the function setRewardsDuration() the global variable is passed in the event instead of the local one. This is a redundant read from global states.

Library SafeMath is redundant. Since 0.8.0 solidity there are built-in arithmetic checks and SafeMath becomes a rudiment.

04. Withdraw logic

- Informational ⓘ Acknowledged

In getReward() a possible amount of tokens (a minimum of the user's rewards and the reward token balance of the contract) should be transferred in order to prevent possible problems with the last user.

05. Token support

- ■ Informational ⓘ Acknowledged

There is no support of RFI tokens and tokens with commissions; tokens for rewards with tokens for staking should be ordinary, without commissions or changing balances with time.

4.6 HcSwapAvaxRouter

4.6.1 Overview

Router contract for HurricaneSwap. Unlike other Uniswap forks, third-party routers are limited in functionality in HurricaneSwap DEX & bridge projects. The contract provides the functionality to perform operations on DEX pairs (adding liquidity, removing liquidity, and swaps). Operations can be performed only through this contract.

4.6.2 Issues

01. Owner overpower

- ■ Medium ⓘ Acknowledged

The owner can stop all users' swaps on this dex.

Recomendation

The owner of the contract should be a timelock or a DAO-like contract.

02. Token support

- Informational ⓘ Acknowledged

There is no support for tokens with commissions and all users' swaps will fail with these tokens.

03. Not enough events

- Informational ⓘ Acknowledged

Functions setOwner(), setCrossToken(), setBscToAvax() and setOperator() don’t have an appropriate event.

Events are very helpful in checking the owner's actions. Because without them it becomes necessary to check all transactions to this contract to find calls to these functions.

4.7 LPBridgeHub

4.7.1 Overview

BSC-AVAX bridge contract for ERC20 tokens. Contract operates in BSC.

4.7.2 Issues

01. Floating pragma

▪ Low ⓘ Acknowledged

The general recommendation is that pragma should be fixed to the version that you are intending to deploy your contracts with. This helps to avoid deploying using an outdated compiler version and shields from possible bugs in the next solidity releases.

02. Structure duplication

▪ Low ⓘ Acknowledged

Structures Task and ReceiveTask (L:26 and L:33) are fully identical.

03. Lack of events in setter functions

▪ Low ⓘ Acknowledged

No events are emitted when fee, feeReceiver or whiteListLP is changed (L:48, L:52, L:60).

04. DoS with block gas limit

▪ Informational ⓘ Acknowledged

Iterating over tasks_ (L:83 - L:91) may exceed gas limit.

4.8 BridgeToken

4.8.1 Overview

Mintable ERC20 token for BSC-AVAX bridge. Token performs logic on AVAX side and it is minted by LPBridgeRemote.

4.8.2 Issues

01. Floating pragma

▪ Low ⓘ Acknowledged

The general recommendation is that pragma should be fixed to the version that you are intending to deploy your contracts with. This helps to avoid deploying using an outdated compiler version and shields from possible bugs in the next solidity releases.

4.9 HurricaneArbitrageHelper

4.9.1 Overview

AVAX part of the arbitrage mechanism.

4.9.2 Issues

01. Swap's zero slippage

▪ Medium ⓘ Acknowledged

Setting slippage to 0 (L:181, L:192, L:194) makes swaps on DEX vulnerable to a sandwich attack.

02. Unnecessary equality check

▪ Low ✓ Resolved

No need to check the returned boolean on equality to true/false (L:35).

03. Lack of events in set functions

▪ Low ⓘ Acknowledged

No events in setArbitrager(), setRouter(), setFactory(), setHOGWallet(), setWAVAX() functions (L:48-75).

04. Extra calculation

▪ Informational ✓ Resolved

Calculation of the swap deadline (L:176) is unnecessary. Swap transactions will be in the same block, consequently, they will have the same timestamp as the arbitrage transaction.

05. Gas optimizations

- Informational ◎ Partially fixed

a. Unused variable WAVAX_Deprecated.

b. _router, wAVAX, _HOGWallet variables are read multiple times in the doSwap() function.

c. The wAVAX variable is read multiple times in the doArbitrage() function.

d. The _factory variable is read multiple times in the getReserves() function.

e. Excessive computations L:225, simple multiplication inequality may be used as it is done in getArbitrageAmount().

06. Arbitrage can be unrefunded

- Informational ⓘ Acknowledged

DUST (L:164) sets the token's amount lower bound for refund. If the contract has less than DUST tokens, they won't be returned to the arbitrager until the required amount is reached. While performing arbitrage, the arbitrager should consider only the tokens with an appropriate decimals parameter.

Update

In the provided fix were added the requirements that amounts should be greater than DUST to call doSwap():

```
if (amount1In >= DUST) {
    doSwap(token1, token0, amount1In);
}

.
.
.

if (amount0In >= DUST) {
    doSwap(token0, token1, amount0In);
 }
```

However, tokens still stay on the contract`s balance even if they were not swapped. Then tokens are supposed to be returned, but they won`t because of L:162 refund() requirement.

4.10 LPBridgeRemote

4.10.1 Overview

BSC-AVAX bridge contract for ERC20 tokens. Contract operates in AVAX.

4.10.2 Issues

01. Initialized token mapping rewrite

▪ Medium ⓘ Acknowledged

The owner can change the already initialized token mapping (L:55). This may lead to minting (L:100) of multiple target tokens for origin token or to nullifying of the users' balances (L:66, L:86).

Recommendation

It is strongly advised to restrict change of initialized maps in tokenMapping and reverseTokenMapping or to add logic ensuring a correct balances transition.

02. Gas optimization

- Low ⓘ Acknowledged

a. setPause() (L:104) visibility can be changed to external.

03. Floating pragma

- Low ⓘ Acknowledged

The general recommendation is that pragma should be fixed to the version that you are intending to deploy your contracts with. This helps to avoid deploying using an outdated compiler version and shields from possible bugs in the next solidity releases.

04. Lack of events

- Low ⓘ Acknowledged

No events are emitted when fee, feeReceiver, or tokenMapping/reverseTokenMapping is changed (L:43, L:47, L:55).

4.11 HcSwapAvaxPair

4.11.1 Overview

LP token contract for avalanche part. Actions can be performed only by the owner of the HcSwapAvaxFactory contract (by logic it should be the HcSwapAvaxRouter contract).

4.11.2 Issues

01. Not enough events

- Informational ⓘ Acknowledged

The function setCrossPair() doesn't have an appropriate event.

Events are very helpful in checking the owner's actions. Because without them it becomes necessary to check all transactions to this contract to find calls to these functions.

02. View function fail

- Informational ⓘ Acknowledged

Function burnQuery() can fail if there are no LP tokens.

It is a better practice to make view and pure functions that don't fail.

4.12 HcSwapV2Router02

4.12.1 Overview

Router contract for BSC part of HurricaneSwap.

4.12.2 Issues

01. Owner overpower

▪ Medium ⓘ Acknowledged

The owner of the contract has access to LP tokens of the users. Also, they can pause all users' swaps.

Recommendation

The owner of the contract should be a timelock or a DAO-like contract.

02. Not enough events

▪ Low ⓘ Acknowledged

Functions setOwner(), IHcSwapBSCFactory() and setOperator() don't have appropriate events.

Events are very helpful in checking the owner's actions. Because without them it becomes necessary to check all transactions to this contract to find calls to these functions.

03. transfer() function

- Low ⓘ Acknowledged

In the function addLiquidityFromUser() transfer() is used to send native tokens. It is better to use the call() function. If the receiver is a contract and it performs a certain operation when receiving ETH, the call may fail as the transfer() function forwards only 2300 gas. It is better to use the call() function as it forwards all gas.

04. Token support

- Informational ⓘ Acknowledged

There is no support for tokens with commissions and all users' swaps will fail with these tokens.

4.13 HcSwapBSCPair

4.13.1 Overview

LP token contract for BSC part. Actions can be performed only by the owner of the HcSwapFactory contract (logically it should be the HcSwapV2Router02 contract).

4.13.2 Issues

01. Owner overpower

▪ Medium ⓘ Acknowledged

The owner of the contract can mint any amount of LP tokens without transferring underlying tokens to the pair. Also, they have access to LP tokens of other users. Moreover, they have access to all reserves of the pair and can change rates by using this.

Recommendation

The owner of the contract should be a timelock or a DAO-like contract.

4.14 HcToken

4.14.1 Overview

Bridge mintable and burnable ERC20 token with a blacklist for users' transfers.

4.14.2 Issues

01. Owner overpower

▪ Medium ⓘ Acknowledged

The owner of the token can mint any amount of tokens, burn any amount of tokens from any account. Also, they can block transfers of some users.

Recommendation

The owner of the contract should be either a timelock or a multi-sig account instead of an EOA.

02. Gas optimization

- Low Acknowledged

Variable originAddress is redundant. It wastes gas on deployment only.

4.15 PancakeLibrary

4.15.1 Overview

PancakeSwap DEX amounts calculator.

4.15.2 Issues

01. Fees are unconcerned in amounts calculations

- Informational Resolved

Get amount functions' calculations are performed without fees. Their potential usage may lead to incorrect work of dependent functions.

4.16 HcSwapFactory

4.16.1 Overview

Factory contract for BSC part of HurricaneSwap. No issues were found.

4.17 HcSwapERC20

4.17.1 Overview

It is a base contract for HcSwapBSCPair. Contains logic of the ERC20 token. No issues were found.

4.18 Timelock

4.18.1 Overview

Standard Timelock contract by Compound Finance. No issues were found.

4.19 HcSwapAvaxFactory

4.19.1 Overview

Factory contract for avalanche part of HurricaneSwap. No issues were found.

4.20 HcTokenFactory

4.20.1 Overview

Factory contract for bridge tokens. Deploys HcToken contract. No issues were found.

4.21 LPQueue

4.21.1 Overview

Library for bridge tasks queue. No issues were found.

4.22 HcSwapAvaxERC20

4.22.1 Overview

It is a base contract for HcSwapAvaxPair. Contains logic of the ERC20 token. No issues were found.

4.23 HurricaneLibrary

4.23.1 Overview

HurricaneSwap DEX amounts calculator. No issues were found.

4.24 RewardsDistributionRecipient

4.24.1 Overview

This is the support contract for access restriction. Contract StakingRewards inherited from it. No issues were found.

4.25 HcSwapHelper

4.25.1 Overview

Support contract with pure and view functions for performing convenient calculations for DEX. No issues were found.

5. Conclusion

No high or critical severity issues were found. The HurricaneSwap project is highly centralized: Factory and Pair contracts are accessible only for the owner, i.e. Router contract, which has its own owner (EOA by 2021-12-09 for the deployed HcSwapAvaxRouter) and non-public admin list. Users using 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.

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

@hashexbot

blog.hashex.org

linkedin

github

twitter

HashEx
Blockchain Security