#HashEx
BLOCKCHAIN SECURITY

AmbiDex

smart contracts final audit report

August 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 AmbiDex team to perform an audit of their smart contract. The audit was conducted between 12/07/2022 and 26/07/2022.

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.

Some of the audited contracts are designed to be deployed with proxies. Users have no choice but to trust the owners, who can update the contracts at their will.

The code is available at @rekursive-labs/ambidex GitHub repository after 08e9064 commit.

Update: the AmbiDex team has responded to this report. The updated code is located in the same repository after the commit f3f814d.

2.1 Summary

Project name	AmbiDex
URL	http://ambidex.fi
Platform	Hedera
Language	Solidity

2.2 Contracts

Name	Address
AddressProvider	
MasterChef	
LaunchADX	
StakingADX	
Treasury	
VestingADX	
MultiSigWallet	
utils/ folder	
lib/ folder	
dex/ folder	
Interfaces	

3. Found issues

High	2 (9%)
Medium	1 (4%)
Low	16 (70%)
Info	4 (17%)

C1. AddressProvider

ID	Severity	Title	Status
C1-01	High	Excessive owner's rights	Resolved
C1-02	Low	No events	Resolved
C1-03	Info	Variable visibility	Resolved

C2. MasterChef

ID	Severity	Title	Status
C2-01	Low	No events	Resolved
C2-02	Low	Indexation error	Resolved
C2-03	Low	Gas optimizations	Resolved
C2-04	Info	No support of tokens with commisions	Acknowledged

C3. LaunchADX

ID	Severity	Title	Status
C3-01	Low	Gas optimizations	Partially fixed
C3-02	Info	Sale period is not determined	Acknowledged

C4. StakingADX

ID	Severity	Title	Status
C4-01	Low	Gas optimization	Resolved

C5. Treasury

ID	Severity	Title	Status
C5-01	High	Excessive owner's rights	Partially fixed
C5-02	Low	No events	Resolved
C5-03	Low	Gas optimization	Acknowledged

C6. VestingADX

ID	Severity	Title	Status
C6-01	Low	No events	Resolved
C6-02	Low	No view function for vestingSchedulesIds[] array length	Resolved

C6-03	Low	Gas optimizations	Partially fixed

C9. lib/ folder

ID	Severity	Title	Status
C9-01	Low	IHederaTokenService types mismatching	Acknowledged
C9-02	Info	HederaTokenService disabled functionality	Resolved

C10. dex/ folder

ID	Severity	Title	Status
C10-01	Medium	UniswapV2Router wHBAR functionality	Resolved
C10-02	Low	UniswapV2Pair initialization	Acknowledged
C10-03	Low	UniswapV2Router redundant code	Resolved
C10-04	Low	UniswapV2Router associations	Partially fixed
C10-05	Low	UniswapV2Router gas optimization	Resolved

4. Contracts

C1. AddressProvider

Overview

The contract stores `Treasury`, `MasterChef`, `LaunchADX`, `StakingADX`, `VestingADX`, `UniswapV2Factory` addresses, which are acquired by other contracts. The contract was initially designed to be deployed with proxy.

Issues

C1-01 Excessive owner's rights

High Resolved

Other contracts' logic depends on the addresses this contract stores. The owner can change them to malicious ones, which may lead to pernicious consequences including the loss of users' assets.

```
function setTreasury(address treasury) external override onlyOwner {
    require(treasury != address(0), ERROR_ZERO_ADDRESS);
    _addresses[TREASURY] = treasury;
}

function setMasterChef(address masterChef) external override onlyOwner {
    require(masterChef != address(0), ERROR_ZERO_ADDRESS);
    _addresses[MASTERCHEF] = masterChef;
}

function setLaunchADX(address launchADX) external override onlyOwner {
    require(launchADX != address(0), ERROR_ZERO_ADDRESS);
    _addresses[LAUNCH_ADX] = launchADX;
}

function setStakingADX(address stakingADX) external override onlyOwner {
    require(stakingADX != address(0), ERROR_ZERO_ADDRESS);
    _addresses[STAKING_ADX] = stakingADX;
```

```
    }

    function setVestingADX(address vestingADX) external override onlyOwner {
        require(vestingADX != address(0), ERROR_ZERO_ADDRESS);
        _addresses[VESTING_ADX] = vestingADX;
    }

    function setUniswapV2Factory(address uniswapV2Factory) external override onlyOwner {
        require(uniswapV2Factory != address(0), ERROR_ZERO_ADDRESS);
        _addresses[UNISWAP_V2_FACTORY] = uniswapV2Factory;
    }
```

Recommendation

The ownership should be transferred to a Timelock-like contract with a minimum delay of at least 24 hours. This won't stop the owner from possible rights abuse but it will help users to be informed about upcoming changes. Also, consider adding ERC165 standard support to the contracts whose addresses can be updated and check whether a new address implements the interface of the updated contract before address change to avoid accidental address setting.

Update

Within the update stored addresses re-initialization was restricted. Contract's deployment sheme has been changed from proxy to the direct one.

C1-02 No events

Low Resolved

We recommend emitting events on important value changes to be easily tracked off-chain. No events are emitted in setter functions.

C1-03 Variable visibility

Info Resolved

`ERROR_ZERO_ADDRESS` contains an error message for several setter functions. Its public visibility is redundant and can confuse users.

C2. MasterChef

Overview

MasterChef farming contract with a custom bonus reward system dependent on staked LP and `IADX` tokens. The contract is designed to be deployed with proxy.

Issues

C2-01 No events

Low Resolved

We recommend emitting events on important value changes to be easily tracked off-chain. No events are emitted in the `add()` and `set()` functions.

C2-02 Indexation error

Low Resolved

If the `poolExists()` modifier receives on input pool ID greater than `poolInfo.length- 1`, the transaction will be reverted with an inconsistent indexation error message.

C2-03 Gas optimizations

Low Resolved

a. There is no need for zero initializing of the `totalAllocPoint` variable in the `__MasterChef_init_unchained()` function.

b. No need to require `int256(_amount) <= type(int64).max` on L162, 190, since the `_amount` variable has int64 type.

C2-04 No support of tokens with commisions

Info Acknowledged

The contract doesn't support tokens with fees on transfers or rebasing tokens. Creating a liquidity pool with some will end up all LP tokens draining. The owner must avoid adding pools for such tokens.

C3. LaunchADX

Overview

The contract enables private and public `ADX` token sales. The private sale is only available for whitelisted users. Both sales have a cap for `ADX` token sold. After tokens are bought with a private sale, they will be locked in the `VestingADX` contract, in case of a public sale they are instantly transferred to a user. The contract is designed to be deployed with proxy.

Issues

C3-01 Gas optimizations

Low Partially fixed

a. `addressProvider` and `privateSaleAmount` /`publicSaleAmount` state variables are read multiple times in `buyADXPrivate()`/`buyADXPublic()` ;

b. Tokens addresses and their decimals parameters received with `tokenUSDC()`, `tokenADX()`, and `decimals()` external calls can be memorized to storage variables to reduce gas consumption;

c. Unchecked math could be used in L58, 78.

C3-02 Sale period is not determined

Info Acknowledged

The owner can start and stop a private/public sale any time they want to. Though, the sale can not be re-opened after it's closed.

C4. StakingADX

Overview

Staking for the `ADX` token. Rewards are earned in `IADX` token. The code structure is quite similar to the `MasterChef` contract. The contract is designed to be deployed with proxy.

Issues

C4-01 Gas optimization

Low Resolved

In the `updateReward()` modifier, state variable `_rewardPerTokenStaked` is read after writing to storage.

C5. Treasury

Overview

The contract is responsible for creating `ADX` and `IADX` tokens and their further distribution. `ADX` is minted with an initial supply of 25'000'000 tokens without any extra mint possibility, whereas `IADX` starts with 0 initial supply and can be minted by `StakingADX` and `MasterChef`. The contract is designed to be deployed with proxy.

Issues

C5-01 Excessive owner's rights

High Partially fixed

a. The owner can withdraw all `ADX` tokens with `transferToken()` and `withdrawADX()` functions;

b. The owner can substitute the `StakingADX` or `MasterChef` address to their own and unlimitedly mint `IADX` tokens, see more at Excessive owner's rights issue in the AddressProvider section.

```
    function withdrawADX(address to, int64 amount) external override nonReentrant {
        ...
        require(
            _msgSender() == owner() || _msgSender() == launchADX || _msgSender() ==
vestingADX,
            "Treasury: only owner/launch/vesting" );
        IERC20(tokenADX).safeTransfer(to, uint64(amount));
        ...
    }

    function transferToken( address tokenAddr, int64 amount, address to
    ) external override onlyOwner nonReentrant {
        IERC20(tokenAddr).safeTransfer(to, uint64(amount));
    }
```

Recommendation

a. The ownership should be transferred to a Timelock-like contract with a minimum delay of at least 24 hours and MultiSig account as admin. This won't stop the owner from possible rights abuse but it will help users to be informed about upcoming changes.

b. See C1 recommendation.

Update

a. The owner was removed from possible `withdrawADX()` callers, but still can abuse `transferToken()` to drain `ADX`.

b. See C1 update.

C5-02 No events

Low Resolved

We recommend emitting events on important value changes to be easily tracked off-chain. No events are emitted in `onlyOwner` functions.

C5-03 Gas optimization

Low Acknowledged

a. In function `createADXToken()` gloval variable `tokenADX` is read after writing. The same in function `createIADXToken()` with variable `tokenIADX`.

b. In function `transferADXToBuyer()` global variables `addressProvider` and `availableADXForBuyers` are read multiple times. The same in function `mintIADX()` with variables `addressProvider` and `tokenIADX`.

C6. VestingADX

Overview

The contract holds tokens bought in private sale with a total vesting period of 1.5 years. The contract is designed to be deployed with proxy.

Issues

C6-01 No events

Low Resolved

We recommend emitting events on important value changes to be easily tracked off-chain. No events are emitted in the `depositADX()` function.

C6-02 No view function for vestingSchedulesIds[] array length

Low Resolved

There is no public view function returning the length of the `vestingSchedulesIds[]` array. Lack of information about total number of vestings may be problematic for users trying to analyze the project's state of affairs or for those who monitor vesting updates.

C6-03 Gas optimizations

Low Partially fixed

a. In the function `withdraw()` in line 69 the variable `vestingSchedule` should be read into the memory instead of the storage to reduce the number of reads of fields in this structure;

b. In the function `depositADX()` the state variable `addressProvider` is read multiple times;

c. There is no need for the `SafeMath` library in the 0.8.0 and above version of solidity. It is embedded into the compiler;

d. In function `withdraw()` on line 80 in storage all struct is written, but only one field is changed.

C7. MultiSigWallet

Overview

Multi signature wallet by Gnosis with minor changes due to compiler version update. No issues were found.

C8. utils/ folder

Overview

The folder contains generally imported contracts and libraries. Among them: math libraries, proxy contract, SafeERC20 wrapper, execution context providers, address library, SafeCast module, storage read auxiliaries, and hedera token helper. Most of the contents are imported from the OpenZeppelin repository.

HTSHelper

A modified version of the TokenCreateContract from @hashgraph/hedera-smart-contracts Hedera GiHub repository. Implements helper functions for token creation, minting, and burning. No issues were found.

C9. lib/ folder

Overview

The folder with Hedera Smart Contract Service supporting files. Imported from the @hashgraph/hedera-smart-contracts Hedera GiHub repository.

Issues

C9-01 IHederaTokenService types mismatching

Low Acknowledged

Response codes' types are changed from int64 to int256 according to the official Hedera Token Service contract.

C9-02 HederaTokenService disabled functionality

Info Resolved

Default renew period `defaultAutoRenewPeriod` is disabled according to the official Hedera Token Service contract.

C10. dex/ folder

Overview

The folder includes UniswapV2 protocol contracts adapted for Hedera service.

Issues

C10-01 UniswapV2Router wHBAR functionality

Medium Resolved

No functions `unwrap()` and `wrap()` for `wHBAR`. This issue may cause problems in arbitrage for pairs with `wHBAR`. Also, users, who called swap to `wHBAR` by mistake, have no simple option to convert it to `HBAR`.

Issue was introduced in the code update.

Recommendation

This issue can be fixed in two ways:

1. Make functions `unwrap()` and `wrap()` in UniswapV2Router contract;
2. Make separate contract for `wHBAR` with functionality of a token and with `unwrap()` and `wrap()` functions.

The first way is easier than the second one, but the second way is better in terms of backwards compatibility.

C10-02 UniswapV2Pair initialization

Low Acknowledged

`UniswapV2Pair` contract uses `HTSHelper.createToken()` to create a token in Hedera Token Service. The newly created Pair contract becomes its own ADMIN, MINTER, and PAUSER, but the pausable and updating methods aren't implemented. Also, it uses `DELEGATABLE_CONTRACT_ID_KEY` key type, though the Pair contract is not a proxy.

```
    constructor() payable {
        ...
```

```
        lpToken = createToken("Ambidex Liquidity Token", "ADXLT", 0, 6, address(this));
    }

    function createToken(
        string memory name,
        string memory symbol,
        uint256 initialSupply,
        uint256 decimals,
        address treasury
    ) internal returns (address createdTokenAddress) {
        ...
        keys[0] = getSingleKey(
            HederaTokenService.ADMIN_KEY_TYPE,
            KeyHelper.DELEGATABLE_CONTRACT_ID_KEY,
            treasury
        );
        keys[1] = getSingleKey(supplyPauseKeyType, KeyHelper.DELEGATABLE_CONTRACT_ID_KEY,
treasury);
        IHederaTokenService.HederaToken memory myToken = IHederaTokenService.HederaToken(
            name,
            symbol,
            treasury,
            "",
            false,
            0,
            false,
            keys,
            getSecondExpiry(uint32(block.timestamp + 10 * 365 * 24 * 60 * 60)) // 10 years
        );
        ...
    }
```

C10-03 UniswapV2Router redundant code

Low Resolved

`addLiquidity()` contains a requirement of `pair != address(0)` on L93, which has no use since `UniswapV2Library.pairFor()` never returns zero.

C10-04 UniswapV2Router associations

Low Partially fixed

In swap functions, Hedera Token Service associations can be made for users, who call the function, to make the execution possible for new users without a corresponding token association. Similarly, it was done in `addLiquidity()` function in L125-130.

C10-05 UniswapV2Router gas optimization

Low Resolved

The global variable `wHBAR` can be set as immutable. Its creation could be moved to the constructor section.

Issue was introduced in the code update.

C11. Interfaces

Overview

It aggregates all inspected project's interfaces. Including: `IAddressProvider.sol`, `IMasterChef.sol`, `ILaunchADX.sol`, `IStakingADX.sol`, `ITreasury.sol`, `IVestingADX.sol`, `IMultiSigWallet.sol`. No issues were found.

5. Conclusion

2 high, 1 medium, 16 low severity issues were found during the audit. 1 high, 1 medium, 10 low issues were resolved in the update.

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

Furthermore, such sensitive tokenomics contracts as `VestingADX`, `Treasury`, `LaunchADX`, `StakingADX`, `MasterChef` are designed to be deployed with proxies, opening the possibility for their implementation change over time that may have malicious intent or cause new vulnerabilities appearance.

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