YZ

smart contracts
final audit report

March 2023

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 YZ team to perform an audit of their smart contract. The audit was conducted between 2023-01-13 and 2023-01-18.

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 @Fintal78/Charged-token Github repository and was audited after the 0a8ce77 commit.

Update. A recheck was done after the commit 7abab8f.

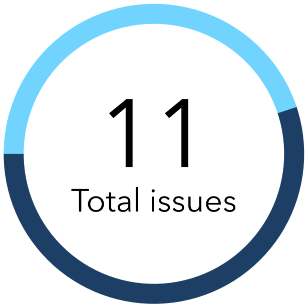
2.1 Summary

Project name	YZ
URL	http://yz-network.com/
Platform	Polygon Network
Language	Solidity

2.2 Contracts

Name	Address
InterfaceProjectToken	
LiquidityToken	
ProjectToken	
DelegableToLT	
ContractsDirectory	
AddressSet & StringSet	
General recommendations	

3. Found issues



●	Low	5 (45%)
●	Info	6 (55%)

C1. InterfaceProjectToken

ID	Severity	Title	Status
C1-01	Low	Possible wrong returning value	Acknowledged
C1-02	Low	Gas optimizations	Resolved

C2. LiquidityToken

ID	Severity	Title	Status
C2-01	Low	Gas optimizations	Partially fixed
C2-02	Info	Fees up to 100%	Acknowledged

C5. ContractsDirectory

ID	Severity	Title	Status
C5-01	● Info	Overcomplicating code	✓ Acknowledged

C6. AddressSet & StringSet

ID	Severity	Title	Status
C6-01	● Low	Lack of getters	✓ Acknowledged
C6-02	● Low	Gas optimizations	✓ Acknowledged

C7. General recommendations

ID	Severity	Title	Status
C7-01	● Info	Lack of automated testing	✓ Acknowledged
C7-02	● Info	Lack of documentation	✓ Acknowledged
C7-03	● Info	Lack of events	✓ Resolved
C7-04	● Info	Importing from OpenZeppelin	✓ Acknowledged

4. Contracts

C1. InterfaceProjectToken

Overview

A user-interacting contract to manage the restricted functions of LiquidityToken and ProjectToken contracts.

Issues

C1-01 Possible wrong returning value ● Low ✓ Acknowledged

In the `getValueProjectTokenPerVestingSchedule()` function, there's should be a check of `(_blockTime - _dateStart) / _durationLinearVesting <= 1`, otherwise, it would return linearly increasing value over time.

Recommendation

Limit the return value from above or add the NatSpec function description with a justification of the current function behavior.

Team response

The team responded that this function is declared public to easily be accessible to an off-chain monitoring application. Also a comment just before the function was added to provide the usage limitations of the function

C1-02 Gas optimizations ● Low ✓ Resolved

1. The `dateLaunch` and `dateEndCliff` variables are read multiple times from storage in the `setStart()` function.
2. The `dateLaunch` variable is read multiple times from storage in the `claimProjectToken()` function.

3. The `dateEndCliff` and `valueProjectTokenToFullRecharge[_user]` variables are read multiple times from storage in the `getValueProjectTokenVestedAndUpdateLTBalances()` function.

C2. LiquidityToken

Overview

Implementation of the ERC-20 token standard built on top of OpenZeppelin implementation. Supports 2 types of vesting schedules: cliff and linear vesting. Minting, burning, and rewards claiming are managed by the InterfaceProjectToken contract.

Centralization risks

Mint is open for owner

1. The project owner can mint an unlimited number of Liqui tokens to users before the project is started, using the function `allocateLTByOwner()`. The check whether the project is started is made by calling the Interface Token:

```
function checkProjectNotYetLaunched() private view {
  if (address(interfaceProjectToken) != address(0x0)) {

    uint _dateLaunch = interfaceProjectToken.dateLaunch();

    if (_dateLaunch > 0) {
      require(
        block.timestamp < _dateLaunch,
        "Project already launched");
    }
  }
}
```

The contract owner can change the address of the interfaceProjectToken.

```
function setInterfaceProjectToken(InterfaceProjectToken _interfaceProjectToken)
  public onlyOwner() {

  require(
    address(_interfaceProjectToken) != address(0x0),
    "Set a valid address");

  interfaceProjectToken = _interfaceProjectToken;
}
```

If the contract owner's account is compromised even if the project has already started an attacker can change the address of the Interface Project token to a malicious one that will return a zero launch date. The launch date check would be passed and the attacker would have the possibility to mint an arbitrary number of tokens to his addresses.

2. The owner can set a big amount of rewards in tokens with the `setStakingRewards()` function and in the next block frontrun transactions to be the first who gets and sells tokens.

Recommendation

1. Do not allow changing the Interface Project token address once it is set.2. Limit the maximum staking rewards.

Put the contract's owner behind a 24h minimum Timelock contract and secure the owner's account with a multisig.

Team response

In order to find the best compromise between security and flexibility, a function called lockInterfaceProjectToken was added which can lock the InterfaceProjectToken when the owner is sure that the Project Token will not have to be modified in the future.

To further secure the function allocateLTByOwner in case the owner's account gets compromised :

- a maximum token amount (maxInitialTokenAllocation) has been set up in the constructor that can not be exceeded in terms of token allocation (outside of staking rewards)

- the function terminateAllocations was added to terminate the allocation of Charged Tokens as soon as all the allocations have been done for the given investment round.

Also, a limitation was set up on the max amount of tokens (maxStakingTokenAmount) and the APR (maxStakingAPR) for a staking campaign, defined by the Project Owner at the deployment of the Charged Token (a max token amount is not sufficient as an attacker could be harmful with the max amount of tokens allocated during a very short period of time).

Issues

C2-01 Gas optimizations ● Low ✓ Partially fixed

1. The `maxWithdrawFeesPerThousandForLT` and `maxClaimFeesPerThousandForPT` variables should be declared as immutable.
2. The `currentRewardPerShare1e18` variable is read multiple times in the `updateStakingParameters()` and `updateStakingAndGetHodlRewards()` functions.
3. The `stakingDateLastCheckpoint` variable is read multiple times from storage in the `getUpdatedRewardPerShare1e18()` function.
4. The `fullyChargedBalance` variable is read multiple times from storage in the `dischargeUserTokens()` function
5. `currentRewardPerShare1e18` variable are read multiple times in the `updateStakingParameters()` and `updateStakingAndGetHodlRewards()` functions

Update

Points 1, 2, and the first part of 5 were fixed in the update.

C2-02 Fees up to 100% ● Info ✓ Acknowledged

The constructor section contains the initialization of fees limiters `maxWithdrawFeesPerThousandForLT` and `maxClaimFeesPerThousandForPT`. However, these limits from above can be set up to 100%:

```
  require(
    _maxWithdrawFeesPerThousandForLT < multiplier1K,
    "Maximum withdrawal fee must be lower than 100%");

  require(
    _maxClaimFeesPerThousandForPT < multiplier1K,
    "Maximum claim fee must be lower than 100%");

  uint public constant multiplier1K = 1000;
```

C3. ProjectToken

Overview

Implementation of ERC-20 token standard built on top of OpenZeppelin implementation. Supports 2 types of vesting schedules: cliff and linear vesting. Minting and burning are accessible to addresses from an owner-controlled whitelist.

C4. DelegableToLT

Overview

Implementation of the ERC-20 token standard built on top of OpenZeppelin implementation. Part of ProjectToken's inheritance scheme.

Centralization risks

Mint is open for the owner

The `mintByInterfaceProjectToken()` is restricted for the `validatedInterfaceProjectToken` set of addresses, which is directly modifiable by the project owner.

```
function mintByInterfaceProjectToken(address _user, uint _value)
```

```
      public onlyInterfaceProjectToken() {
      _mint(_user, _value);
    }

    modifier onlyInterfaceProjectToken() {
      ensureOnlyInterfaceProjectToken();
      _;
    }

    function ensureOnlyInterfaceProjectToken() private view {
      require(
        validatedInterfaceProjectToken.contains(msg.sender),
        "Only validated InterfaceProjectToken"
        );
    }

    function addInterfaceProjectToken(address _interfaceProjectToken)
      public onlyOwner() {
      validatedInterfaceProjectToken.store(_interfaceProjectToken);
    }
```

Recommendation

The ownership of the DelegableToLT (ProjectToken) contracts must be transferred to secured accounts, e.g. Timelock with MultiSig admin.

Team response

To reduce the risks, the function finalizeListOfValidatedInterfaceProjectToken was added to stop the addition of new InterfaceProjectToken Contracts once the owner has deployed all the needed Contracts.

C5. ContractsDirectory

Overview

A contract storing the registry of projects, owners, and their relationships.

Issues

C5-01 Overcomplicating code ● Info ✓ Acknowledged

Changing the project name requires at least 3 transactions from the owner. Foregoing any of them will result in contract misconfiguration.

```
    // Change Project Name, this requires several transactions
    // => First disable user interactions with all LT contracts related to the Project
subject to changes

    function changeProjectName(string memory _oldProjectName, string memory
_newProjectName) public onlyOwner {
        projects.remove(_oldProjectName);
        projects.store(_newProjectName);
    }

    // after changing the name of a Project, all related LT must be manually associated to
the new Project name
    function allocateLTToProject(address _contract, string memory _project) public
onlyOwner {
        projectRelatedToLT[_contract] = _project;
    }

    // after changing the name of a Project, its owner must be associated to the new
Project name
    function allocateProjectOwnerToProject(address _projectOwner, string memory _project)
public onlyOwner {
        whitelist[_projectOwner] = _project;
    }
```

Recommendation

We recommend merging this code into a single function to reduce the possibility of accidental or intentional mistakes.

Team response

Merging the code in a single function would complexify the smart contract as it requires to identify all the Charged Tokens related to a given Project, which would require adding an array, a loop... with additional gas costs. All this for something that might happen very rarely, and even if it does, the Project Owner might want to keep the old name in the ContractsDirectory (similarly in case of rebranding, the Projects don't change the ticker/symbol of their token). Therefore a compromise was chosen between simplicity/efficiency and flexibility. If needed later on, the process could be automated offchain, keeping the smart contract simple without additional variable, loop etc…).

Furthermore, even if a mistake is done by the Owner :

- it would impact only the ContractsDirectory which is just a database to have an overview of the projects and Charged Tokens deployed, it would not affect any functionality of the Charged Tokens and Project Tokens

- it can be corrected at any time by calling the functions again

C6. AddressSet & StringSet

Overview

Implementations of the EnumerableSet library for addresses and strings.

Issues

C6-01 Lack of getters ● Low ✓ Acknowledged

There are no view functions for getting the set length and its values. We recommend implementing it or adding documentation about the necessity of adding such functionality to derived contracts.

C6-02 Gas optimizations ● Low ✓ Acknowledged

1. `store()` functions of both AddressSet and StringSet contracts read `self.count` variable from storage 4 times.
2. The StringSet contract should store its values in `bytes32` form of hashed strings instead of direct storage. Without a value getter there's no point in storing the full string instead of cheaper hashes.

C7. General recommendations

Overview

This section contains recommendations applying to all audited contracts.

Issues

C7-01 Lack of automated testing ● Info ✓ Acknowledged

Automated tests are written only for library contracts. It is crucially important to have total test coverage for all contracts to ensure that everything works as expected and avoid errors.

Recomendations

Write unit tests for the rest of the contracts.

C7-02 Lack of documentation ● Info ✓ Acknowledged

The project has no in-code documentation. We recommend writing NatSpec documentation for all public and external functions. This will make the code easier to understand and maintain. Also, this documentation will improve user experience as the NatSpec documentation of verified contracts is shown on block explorers.

C7-03 Lack of events ● Info ✓ Resolved

Most of the functions don't emit any events (especially governance ones), complicating the off-chain tracking of important changes.

C7-04 Importing from OpenZeppelin ● Info ✓ Acknowledged

We recommend always sticking to the release versions of contracts imported from the OZ library.

5. Conclusion

5 low severity issues were found during the audit. 1 low issue was resolved in the update.

The audited 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. See the centralization risks chapters for more details.

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