#HashEx
BLOCKCHAIN SECURITY

LFMT Community Token

smart contracts
final audit report

August 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 LFMT team to perform an audit of their smart contract. The audit was conducted between 17/08/2023 and 20/08/2023.

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 directly in .sol file. The SHA-1 hashes of audited contracts:

ERC20LockUpgradableV1_flattened.sol `1197b966ec3b6935f0a8a38dda03354c591140d9`

ProxyAdminUpgrade.sol `dbe5f9e4ba4fe80716f6aa2f2307ccf57aa7b28f`

TransparentUpgradeableProxy.sol `cb923036db61176b9f98c53cf9f6797d508d2cbf`

The updated contracts have the following SHA-1 hashes:

ERC20LockUpgradableV1_flattened.sol `f80d1431d9512ea948f3b41693b9c2f7eb56440e`

ProxyAdminUpgrade.sol `090cd0d8d91b20a1c6741e13250d7ab736bc9aa6`

TransparentUpgradeableProxy.sol `3b199cd5a338754c9e9c802cf1708b8a4b185037`

2.1 Summary

Project name	LFMT Community Token
URL	https://lfmet.com
Platform	Binance Smart Chain
Language	Solidity

2.2 Contracts

Name	Address
ERC20UpgradableV1	
ERC20Upgradeable	
BlackListUpgradeable	
WhiteListUpgradeable	
ManagerUpgradeable	
OpenZeppelin imports	
Multiple contracts	

3. Found issues

High	3 (27%)
Medium	1 (9%)
Low	4 (36%)
Info	3 (28%)

C2. ERC20Upgradeable

ID	Severity	Title	Status
C2-01	High	Locked funds	Resolved
C2-02	High	Balances discrepancy	Resolved
C2-03	Medium	Logic error in vesting	Resolved
C2-04	Low	Decimals discrepancy	Resolved
C2-05	Low	Forced update of referral	Resolved
C2-06	Low	Gas optimizations	Partially fixed
C2-07	Info	Code readability	Resolved
C2-08	Info	Lack of events	Resolved

C3. BlackListUpgradeable

ID	Severity	Title	Status
C3-01	Info	Typos	Resolved

C7. Multiple contracts

ID	Severity	Title	Status
C7-01	High	Initialization problem	Resolved
C7-02	Low	Lack of test and documentation	Partially fixed

4. Contracts

C1. ERC20UpgradableV1

Overview

An ERC-20 standard token with privileged mint and burn functions. Inherits white- and blacklist for transfers, sale for USDT, and gradual vesting of the purchased tokens.

Designed to be deployed with a proxy contract.

Centralization risks

The token is extremely centralized. The owner can:

- set managers to mint tokens without restrictions or burn tokens from an arbitrary user without approval

- include user addresses to blacklist or enable whitelist for all transfers

- update payment token at any moment

- upgrade token contract logic behind the proxy

C2. ERC20Upgradeable

Overview

An ERC-20 standard token with sale for USDT, and gradual vesting of the purchased tokens. Inherits white- and blacklist for transfers.

Designed to be deployed with a proxy contract.

Issues

C2-01 Locked funds

High Resolved

The `exchange()` function accepts pre-defined ERC-20 payments to mint LFMT tokens, but there's no function to withdraw collected funds or exhange them back.

Recommendation

Implement owner governed function to withdraw USDT tokens.

C2-02 Balances discrepancy

High Resolved

The `exchange()` function mint purchased tokens to the token contract's address and locks it in for the user. If purchased amount is greater than `inviteRewardMinExchangeAmount` threshold, the referral percent is locked in for the user, who invited the msg.sender, but this referral share is not minted, i.e. total supply and contract's balance is not updated. This leads to the point of math underflow then contract's balance would be not enough to release user's tokens.

```
function exchange(address inviterAddress, uint256 usdtAmount) public {
  ...
  _mint(address(this), tokenAmount);
  assignLockToken(sender, tokenAmount);

  if (exchangeAmountPerUser[sender] > inviteRewardMinExchangeAmount) {
    assignLockToken(parent, currentReward);
  }
}
```

Recommendation

Mint the `currentReward` amount.

```
...
assignLockToken(sender, tokenAmount);
uint256 currentReward;
```

```
  if (exchangeAmountPerUser[sender] > inviteRewardMinExchangeAmount) {
    ...
  }
  _mint(address(this), tokenAmount + currentReward);
```

C2-03 Logic error in vesting

Medium Resolved

The `vestedAmount()` sums up `lockBalanceOf[beneficiaryAddress]` and `lockErc20Released[beneficiaryAddress]`. Since the `release()` function increases the latter, the vestedAmount's return value is also increased with each successful release. This results in total locking period being reduced for users, who releases their tokens as soon as possible (compared to the ones who releases only once in full).

```
    function release() public {
        address beneficiaryAddress = msg.sender;

        uint256 releaseAmountByPeriodNow = totalReleaseAmountByPeriod(beneficiaryAddress);
        uint256 linearReleaseAmount = vestedAmount(block.timestamp, beneficiaryAddress);

        if (releaseAmountByPeriodNow <= linearReleaseAmount)
        {
            uint256 releasable = releaseAmountByPeriodNow -
lockErc20Released[beneficiaryAddress];
            if (releasable > 0) {
                lockErc20Released[beneficiaryAddress] += releasable;
                emit LockERC20ReleasedEvent(beneficiaryAddress, address(this),
releasable);
                _transfer(address(this), beneficiaryAddress, releasable);
                withdrawAmount[beneficiaryAddress] = withdrawAmount[beneficiaryAddress] +
releasable;
            }
        }
    }

    function vestedAmount(uint256 timestamp, address beneficiaryAddress) public view
returns (uint256) {
        uint256 totalAllocation = lockBalanceOf[beneficiaryAddress] +
lockErc20Released[beneficiaryAddress];
```

```
        if (lockStart == 0 || timestamp < lockStart) {
            return 0;
        } else if (timestamp > lockStart + lockDuration) {
            return totalAllocation;
        } else {
            return (totalAllocation * (timestamp - lockStart)) / lockDuration;
        }
    }
```

Recommendation

Fix the contract's logic according to the documentation and add tests.

C2-04 Decimals discrepancy

Low Resolved

The min and max exchange amounts are initialized as `100e(decimals)` and `1e9 * 1e(decimals)` but the `exchange()` function compares them to `usdtAmount` nominated in payment token decimals. Also, the `setUsdtToken()` function recalculates only the `vipMinExchangeAmount` variable, leaving `minExchangeAmount` and `maxExchangeAmount` untouched and potentially mismatched in decimals.

This issue was introduced in the updated code.

C2-05 Forced update of referral

Low Resolved

The `setInviterRelation()` function forcefully changes the referral address but doesn't update referral count if previous address was not a zero one.

```
    function setInviterRelation(address inviterAddress, address currentUser) external
onlyManagers {
        if (invitation[currentUser] == address(0)) {
            inviteCount[inviterAddress] = inviteCount[inviterAddress] + 1;
        }
        invitation[currentUser] = inviterAddress;
    }
```

C2-06 Gas optimizations

Low Partially fixed

1. Unnecessary reads from storage in the `exchange()` function: `exchangeAmountPerUser[sender]`, `inviteRewardToParentAmount[sender]` variables.

2. Multiple reads from storage in the `withdrawProfit()` function: `lockStart`, `gameStartReleaseBalanceOf[beneficiaryAddress]` variables.

3. Multiple reads from storage in the `vestedAmount()` function: `lockStart`, `lockDuration`.

4. Multiple reads from storage in the `releaseAmountByPeriod()`, `getCurrentRoundEndTime()`, `totalReleaseAmountByPeriod()` functions: `lockPeriodCount`.

5. Multiple reads from storage in the `getCurrentRoundEndTime()` function: `lockTimePerPeriod`.

6. The `lockErc20Released[beneficiaryAddress]` variable should be set to `releaseAmountByPeriodNow` in the `release()` function (instead of being increased by `releasable`).

C2-07 Code readability

Info Resolved

The enum type should be used for user and price types to increase code readability.

C2-08 Lack of events

Info Resolved

The functions `setLockStart()`, `clearLockStart()`, `setVipAddress()`, `setUsdtToken()`, `setMaxExchangeAmount()`, `setAirdropAddress()`, `setLockTimePeriodAndPeriodCount()` don't emit events, which complicates tracking of important off-chain changes.

C3. BlackListUpgradeable

Overview

A simple blacklist contract managed by managers defined in the ManagerUpgradeable contract.

Designed to be deployed with a proxy contract.

Issues

C3-01 Typos

Info Resolved

Typos reduce code readability. Typos in `backlisted`, `blocklisted`.

C4. WhiteListUpgradeable

Overview

A simple whittelist contract managed by managers defined in the ManagerUpgradeable contract.

Designed to be deployed with a proxy contract.

C5. ManagerUpgradeable

Overview

A governance contract which defines a list of privileged manager addresses. List members is updateable by the contract owner.

Designed to be deployed with a proxy contract.

C6. OpenZeppelin imports

Overview

The IERC20, AddressUpgradeable, Initializable, ContextUpgradeable, OwnableUpgradeable, IERC20Upgradeable, IERC20MetadataUpgradeable, and ERC20BurnableUpgradeable contracts are imported from the OpenZeppelin repository without modifications.

C7. Multiple contracts

Overview

This section is for the issues related to multiple audited contracts.

Issues

C7-01 Initialization problem

High Resolved

The inheritance scheme is as follows: ERC20UpgradableV1 is ERC20Upgradeable is BlackListUpgradeable, WhiteListUpgradeable is ManagerUpgradeable is ManagerUpgradeable is OwnableUpgradeable is Initializable.

The Initializable contract from OpenZeppelin library is designed to eliminate nested calls with the `initializer` modifier. Any internal initialization should be marked with the `onlyInitializing` modifier instead.

The `ManagerUpgradeable.__Manager_init()`, `BlackListUpgradeable.__BlackList_init()`, and `WhiteListUpgradeable.__WhiteList_init()` functions are called inside the `ERC20UpgradableV1.initialize()` and have `initializer` modifier. Thus, any attemt to initialize the token contract will fail with 'Initializable: contract is already initialized' error.

Recommendation

Use `onlyInitializing` modifier and add tests.

C7-02 Lack of test and documentation

Low Partially fixed

The project was provided without tests and documentation. We strongly recommend add tests with at least 90% coverage to ensure contract's business logic.

5. Conclusion

3 high, 1 medium, 4 low severity issues were found during the audit. 3 high, 1 medium, 2 low issues were resolved in the update.

The reviewed contracts are very highly dependent on the owner's account. See the centralization risks chapters.

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.

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