#HashEx
BLOCKCHAIN SECURITY

lilAI

smart contracts final audit report

November 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 lilAI team to perform an audit of their smart contract. The audit was conducted between 14/11/2023 and 15/11/2023.

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 is available in the Arbitrum network at 0x655a6BeEBF2361A19549a99486FF65f709BD2646.

2.1 Summary

Project name	lilAI
URL	https://lilai.co
Platform	Arbitrum Network
Language	Solidity
Centralization level	Low
Centralization risk	Low

2.2 Contracts

Name	Address
LILAIERC20	0x655a6BeEBF2361A19549a99486FF65f709BD2646

3. Found issues

Low	4 (80%)
Info	1 (20%)

C4d. LILAIERC20

ID	Severity	Title	Status
C4dIe8	Low	Gas optimizations	Open
C4dIea	Low	Lack of events	Open
C4dIeb	Low	Lack of error messages	Open
C4dIec	Low	Abuse of allowance update	Open
C4dIe9	Info	Usage of default visibility for state variables	Open

4. Contracts

C4d. LILAIERC20

Overview

An ERC20 standard token with fixed initial supply and no extensions.

Issues

C4dIe8 Gas optimizations

Low Open

The `totalSupply_` variable should be declared as immutable to reduce gas costs on reading operations.

```
uint256 totalSupply_;

constructor(uint256 total) {
      totalSupply_ = total;
      balances[msg.sender] = totalSupply_;
}
```

C4dIea Lack of events

Low Open

The constructor section updates balance of the deployer, but no `Transfer()` event is emitted. Lack of standard events may cause external token explorers to display misleading holder's balances.

C4dIeb Lack of error messages

Low Open

There are no error messages in the `transfer()` and `transferFrom()` functions, specifically in requirements for input amounts. Reverting without a reason is discouraged since it doesn't help user to choose right parameters and may provoke an unnecessary gas spending

on obviously reverting transaction.

```
function transfer(address receiver, uint numTokens) public returns (bool) {
        require(numTokens <= balances[msg.sender]);
        ...
}

function transferFrom(address owner, address buyer, uint numTokens) public returns (bool)
{
        require(numTokens <= balances[owner]);
        require(numTokens <= allowed[owner][msg.sender]);
        ...
}
```

C4dlec Abuse of allowance update

Low Open

Credits to OpenZeppelin documentation:

Beware that changing an allowance with `approve()` method brings the risk that someone may use both the old and the new allowance by unfortunate transaction ordering. One possible solution to mitigate this race condition is to first reduce the spender's allowance to 0 and set the desired value afterwards.

C4dle9 Usage of default visibility for state variables

Info Open

The smart contract contains several state variables (`balances`, `allowed`, `totalSupply_`) that use the default visibility. Default visibility for state variables is considered `internal`. While `internal` variables can't be read from external contracts or web3 calls, they can be accessed and modified by derived contracts. It's a best practice to explicitly declare the visibility of state variables to make the code clearer and avoid potential unintended access.

5. Conclusion

4 low severity issues were found during the audit. No issues were resolved in the update.

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. Issue status description

Resolved. The issue has been completely fixed.

Partially fixed. Parts of the issue have been fixed but the issue is not completely resolved.

Acknowledged. The team has been notified of the issue, no action has been taken.

Open. The issue remains unresolved.

Appendix C. 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 D. Centralization risks classification

Centralization level

- **High.** The project owners can manipulate user's funds, lock user's funds on their will (reversible or irreversible), or maliciously update contracts parameters or bytecode.
- **Medium.** The project owners can modify contract's parameters to break some functions of the project contract or contracts, but user's funds remain withdrawable.
- **Low.** The contract is trustless or its governance functions are safe against a malicious owner.

Centralization risk

- **High.** Lost ownership over the project contract or contracts may result in user's losses. Contract's ownership belongs to EOA or EOAs, and their security model is unknown or out of scope.
- **Medium.** Contract's ownership is transferred to a contract with not industry-accepted parameters, or to a contract without an audit. Also includes EOA with a documented security model, which is out of scope.
- **Low.** Contract's ownership is transferred to a well-known or audited contract with industry-accepted parameters.

 contact@hashex.org

 @hashex_manager

 blog.hashex.org

 linkedin

 github

 twitter

#HashEx
BLOCKCHAIN SECURITY