#HashEx
BLOCKCHAIN SECURITY

Mechaversus Mechadium

smart contracts final audit report

September 2022

hashex.org

contact@hashex.org

Contents

1. Disclaimer 3

2. Overview 4

3. Found issues 6

4. Contracts 7

5. Conclusion 10

Appendix A. Issues' severity classification 11

Appendix B. List of examined issue types 12

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 Mechaversus team to perform an audit of their smart contract. The audit was conducted between 20/09/2022 and 22/09/2022.

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.

The code is available at @Mechaversus/bep20-token GitHub repository after the 4709702 commit.

2.1 Summary

Project name	Mechaversus Mechadium
URL	https://mechaversus.co
Platform	Binance Smart Chain
Language	Solidity

2.2 Contracts

Name	Address
IBEP20.sol	

LGEWhitelisted.sol

Mechadium.sol

3. Found issues

Low	3 (75%)
Info	1 (25%)

C2. LGEWhitelisted.sol

ID	Severity	Title	Status
C2-01	Low	Lack of events	Open
C2-02	Low	Usage of msg.sender in a view-function	Open
C2-03	Low	Gas optimization	Open
C2-04	Info	Purchased amount may be accidentally erased	Open

4. Contracts

C1. IBEP20.sol

Overview

IBEP20 interface. No issues were found.

C2. LGEWhitelisted.sol

Overview

Part of `Mechadium` token inheritance scheme. Provide per-user token purchase limitations in specified DEX pair during round periods.

Issues

C2-01 Lack of events

Low Open

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

Recommendation

Emit events in the mentioned functions.

C2-02 Usage of msg.sender in a view-function

Low Open

The function getLGEWhitelistRound() a view-function which calculates the current whitelist round for the caller (`msg.sender` value). This implicates calling this function for a given address off-chain as a default value would be passed for the `msg.sender`.

```
    function getLGEWhitelistRound() public view returns (uint256, uint256, uint256,
```

```
uint256, bool, uint256) {
        if(_lgeTimestamp > 0) {
            uint256 wlCloseTimestampLast = _lgeTimestamp;

            for (uint256 i = 0; i < _lgeWhitelistRounds.length; i++) {
                WhitelistRound storage wlRound = _lgeWhitelistRounds[i];
                wlCloseTimestampLast = wlCloseTimestampLast.add(wlRound.duration);

                if(block.timestamp <= wlCloseTimestampLast)
                    return (i.add(1), wlRound.duration, wlCloseTimestampLast,
wlRound.amountMax, wlRound.addresses[_msgSender()], wlRound.purchased[_msgSender()]);
            }
        }

        return (0, 0, 0, 0, false, 0);
    }
```

Recommendation

Add address as a parameter to the function instead of using `msg.sender` value.

C2-03 Gas optimization

Low Open

a. Multiple storage reads of `wlRound` variable in `getLGEWhitelistRound()` and `_applyLGEWhitelist()`;

b. Multiple storage reads of `_lgeWhitelistRounds[index]` in `modifyLGEWhitelist()`;

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

Recommendation

a, b – memoize repeatedly used state variables to avoid excessive storage reads;

c – remove `SafeMath` import and fix pragma to version 0.8.0 or above.

C2-04 Purchased amount may be accidentally erased

Info Open

If `createLGEWhitelist()` is called after `_lgeTimestamp` is initialised, rounds' `purchased` mappings will be deleted, their correct restoration may take additional time and work. Ensure calling the function, when all set periods passed.

C3. Mechadium.sol

Overview

Implementation of BEP20 interface with a whitelist for token purchases from the set pair. Whitelisted addresses are also limited in max available token amounts to purchase during time periods. No issues were found.

5. Conclusion

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

The audited token realizes the BEP20 interface with constraints on the token purchase from the liquidity pair. No security problems that can menace users' assets were revealed.

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