#HashEx
BLOCKCHAIN SECURITY

GoMining

smart contracts final audit report

October 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 GoMining team to perform an audit of their smart contract. The audit was conducted between 09/10/2023 and 18/10/2023.

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 at gomining2/gmt-contracts/minter-burner Gitlab repository and was audited after the commit 2c20ef9.

The recheck was done after the commit ab4c62c1.

2.1 Summary

Project name	GoMining
URL	http://gomining.com
Platform	Ethereum
Language	Solidity

2.2 Contracts

Name	Address
GoMiningToken	
MinterBurner	
MintReward	
VEGoMiningToken	

3. Found issues

High	1 (8%)
Medium	2 (15%)
Low	5 (38%)
Info	5 (39%)

Ce9. MinterBurner

ID	Severity	Title	Status
Ce9I0f	High	Burn Ratio validation	Resolved
Ce9I0e	Medium	Ownership validation	Resolved
Ce9I11	Low	Lack of events	Acknowledged
Ce9I0d	Low	Gas optimization	Partially fixed
Ce9I10	Info	Explicitly unused mappings	Acknowledged
Ce9I12	Info	Incomplete NatSpec documentation	Resolved

Cea. MintReward

ID	Severity	Title	Status
CeaI27	Medium	Update of veToken address can break reward calculations	Acknowledged
CeaI0b	Low	Gas optimization	Partially fixed
CeaI0c	Info	Creating empty reward distribution	Resolved

Ceb. VEGoMiningToken

ID	Severity	Title	Status
CebI15	Low	Gas optimization	Resolved
CebI14	Low	Incorrect validation	Resolved
CebI13	Info	Fork artifacts	Resolved
CebI28	Info	Not implemented function	Resolved

4. Contracts

Ce8. GoMiningToken

Overview

This contract defines an ERC-20 token implementation. It has mint and burn functionality allowed to the owner.

Also, the contract owner can pause all operations with the contract at any time.

Ce9. MinterBurner

Overview

The contract manages the process of burning existing tokens and minting new tokens as rewards. It includes functionalities to add and remove mint receivers who can receive a portion of minted tokens as rewards.

The contract has a mechanism to define and manage different epochs of burn ratios, which determine the ratio of tokens to be burned for minting rewards

Issues

Ce9I0f Burn Ratio validation

High Resolved

The `addBurnRatioEpoch()` function allows the creation of new burn epochs with specific volume and ratio. However, the function parameters are not validated. This can lead to wrong calculations in the getAmountToMint() function.

```
    function addBurnRatioEpoch(
        uint256 volume,
        uint16 deciRatio
```

```
    ) external onlyRole(CONFIGURATOR_ROLE) {
        burnRatioEpochs.push(BurnRatioEpoch(volume, deciRatio));
    }
```

The same issue in the `setLastBurnRatio()` function.

Recommendation

We highly recommend adding validation for the volume (non-zero) and ratio parameters.

Ce9l0e Ownership validation

Medium Resolved

We highly recommend adding non-zero address validation for the `newOwner` parameter of the `transferTokenOwnership()` function to prevent losing the ownership.

Ce9l11 Lack of events

Low Acknowledged

We recommended emitting events on important value changes to be easily tracked off-chain.

No event are emitted in the `setMintRewardDeciPercent()`, `addMintReceiver()`, `removeMintReceiver()`, `setLastBurnRatio()`, `clearBurnRatioEpochs()` functions.

Ce9l0d Gas optimization

Low Partially fixed

a. The variable `ts` (L18) of the `BurnAndMint` and `ReceiverBurnAndMint` structures is never read in the contract code. Consider removing it or packing it with the `blk` variable by casting types to `uint128`.

b. The `pause()`, `unpause()`, `getMintReceivers()`, `getBurnRatioEpochs()` functions can be declared as `external`.

c. We recommend defining a local variable for loop length (L109, L174) to prevent multiple storage readings in each loop step. For example:

```
const len = burnRatioEpochs.length;
for (uint256 i = 0; i < len; i++) {
    burntSummary += burnRatioEpochs[i].volume;
    if (burntAmount < burntSummary) {
         return (i, burntSummary - burntAmount);
    }
}
```

d. The `require` checks of the `transferTokenOwnership()`, `mintTokens()`, `burnTokens()` are redundant and can be removed, because they already exist in the GoMiningToken functionality.

```
require(IGoMiningToken(Token).owner() == address(this), "MinterBurner: not an owner");
```

Ce9l10 Explicitly unused mappings

● Info ✓ Acknowledged

When the `burnAndMint()` function is executed, the mappings `burnAndMintHistory`, `receiverBurnAndMintHistory`, `receiverBurnAndMintIndex` are updated.

At the same time, these mappings are not explicitly used or read anywhere.

Make sure you need to use them.

Ce9l12 Incomplete NatSpec documentation

● Info ✓ Resolved

The `updateMintReward()` function of the contract does not have documentation. We recommend writing documentation using NatSpec Format. This would help in development, as well as simplify user interaction with the contract (including using the block explorer).

Cea. MintReward

Overview

The contract handles the distribution of rewards to users based on a certain user's balance of VEGoMiningToken.

All rewards are provided by the MinterBurner contract.

Issues

Ceal27 Update of veToken address can break reward calculations

Medium Acknowledged

The contract has a function for updating veToken address.

```
    function updateVeToken(address _veToken) external onlyRole(CONFIGURATOR_ROLE) {
        require(_veToken != address(0), "MintReward: veToken is zero address");
        veToken = _veToken; //@audit can break all reward calculations
    }
```

As rewards are calculated via account balances of veToken on a certain block, updating it may lead to disruptions in reward calculation.

```
    function unclaimedRewards(address _addr) public view returns (uint256) {
        ...

        uint256 amount;
        for (uint256 i = lastRewardIndex + 1; i < rewardCount; i++) {
            Reward memory reward = rewards[i];

            if (IVEGoMiningToken(veToken).totalSupplyAt(reward.blk) != 0) {
                amount += IVEGoMiningToken(veToken).balanceOfAt(_addr, reward.blk) *
reward.amount / IVEGoMiningToken(veToken).totalSupplyAt(reward.blk);
            }
        }
```

```
        return amount;
    }
```

Recommendation

Remove the function to avoid accidentally updating to the wrong token or ensure that the new veToken has the same historical balances as the old one.

Ceal0b Gas optimization

Low Partially fixed

a. The variable `ts` (L18) of the `Reward` structure is never read in the contract code. Consider removing it or packing with the `blk` variable by casting types to `uint128`.

b. We recommend defining state variables `Token` and `veToken` (L21-L22) with `IGoMiningToken` and `IVEGoMiningToken` types respectively. It will allow you not to change type every time on L79, L123, L104, L105 and will save gas on every external call to such addresses.

c. No need to use timestamp and block number (L40-41) in the `ReceivedReward` event. Such metadata already exists in each event and can be easily fetched.

d. Consider using a local variable instead of multiple storage readings of the `veToken` state variable in the for-loop of the `unclaimedRewards()` function.

Ceal0c Creating empty reward distribution

Info Resolved

The `receiveReward()` function allows to create new reward distribution. But during claiming rewards, the `totalSupplyAt()` of vesting tokens is checked (L104). And if the total supply is equal to zero, then such rewards will not be paid to anyone and will simply remain in the contract.Basically, this issue is mitigated by the owner's ability to burn tokens. But we also recommend adding a `totalSupplyAt() != 0` check to the `receiveReward()` function.

Ceb. VEGoMiningToken

Overview

The contract is designed to provide a mechanism for users to lock their tokens for a specified period, thus gaining claiming power in the MintReward contract or gaining voting power in a decentralized governance system. It utilizes epoch-based history and slope calculations to determine users' voting power dynamically based on their token locking behaviors.

This is a Solidity implementation of the CURVE's voting escrow.

Issues

CebI15 Gas optimization

Low Resolved

The `pause()` and `unpause()` functions can be declared as external.

CebI14 Incorrect validation

Low Resolved

The validation on L131 does not match the error message.

```
require(_decimals <= 255, "ve: decimals exceed 18");
```

CebI13 Fork artifacts

Info Resolved

The contract contains artifacts from the original contract:

1. L19 - remove TODO comment;

2. L110 - L111 - unused comments;

3. L312, L316 - unused commented code;

4. L381, L411 - unused TODO comment;

5. L501 - incorrect 'dev' explanation;

CebI28 Not implemented function

Info Resolved

The function `totalSupply(uint timestamp)` is implemented in the original Vyper code, but the Solidity contract lacks it.

5. Conclusion

1 high, 2 medium, 5 low severity issues were found during the audit. 1 high, 1 medium, 2 low issues were resolved in the update.

The reviewed contracts designed to be upgradeable 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.

This audit includes recommendations on code improvement and the prevention of potential attacks.

We recommend covering the found issues with tests after they are fixed.

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

Appendix C. 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.

 contact@hashex.org

 @hashex_manager

 blog.hashex.org

 linkedin

 github

 twitter

#HashEx
BLOCKCHAIN SECURITY