QuillAudits

# Audit Report
# June, 2022

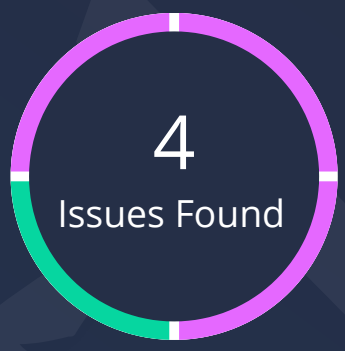
For

GAZECOIN

# Table of Content

# Executive Summary

| | |
|---|---|
| **Project Name** | GazeCoin |
| **Overview** | GazeCoin Token is a simple ERC20 token contract that allows for the mint and burn of tokens. It also inherited the ERC 2771 context contract for meta transactions in order to assign a trustedForwarder to the GazeCoin Token. |
| **Timeline** | 14 June, 2022 to 16 June, 2022 |
| **Method** | Manual Review, Functional Testing, Automated Testing etc. |
| **Scope of Audit** | The scope of this audit was to analyze GazeCoin's smart contract codebases in order to ascertain the quality, security, and correctness. |
| **Fixed in** | *https://github.com/Kratos-Innovation-Labs/GZE_Binance_Smart_Contract/blob/master/contracts/GazeCoin.sol* |
| **Commit Hash** | 19e8a8b167631a6b4034e4643cd936a9fcc866f3 |




High Medium

Low Informational

| | High | Medium | Low | Informational |
|---|---|---|---|---|
| Open Issues | 0 | 0 | 0 | 0 |
| Acknowledged Issues | 0 | 0 | 0 | 0 |
| Partially Resolved Issues | 0 | 0 | 0 | 0 |
| Resolved Issues | 0 | 0 | 1 | 3 |

## Types of Severities

### High

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

### Medium

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

### Low

Low-level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

### Informational

These are severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

## Types of Issues

### Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

### Resolved

These are the issues identified in the initial audit and have been successfully fixed.

### Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

### Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

# Checked Vulnerabilities

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility leve

# Techniques and Methods

Throughout the audit of smart contract, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods and tools were used to review all the smart contracts.

### Structural Analysis

In this step, we have analysed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

### Static Analysis

Static analysis of smart contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

### Code Review / Manual Analysis

Manual analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analysed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

### Gas Consumption

In this step, we have checked the behaviour of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

### Tools and Platforms used for Audit

Remix IDE, Truffle, Truffle Team, Solhint, Mythril, Slither, Solidity statistic analysis.

# Manual Analysis

## A. Contract - GazeCoin.sol

### High Severity Issues

No issues were found

### Medium Severity Issues

No issues were found

### Low Severity Issues

#### A.1 No Check on Constructor for Max Supply Limit

```
  constructor(address _trustedForwarder) ERC2771Context(_trustedForwarder)
ERC20("GazeCoin Metaverse Token", "GZE") {
      _mint(msg.sender, 100000000 ether);
      _owner = msg.sender;
  }
```

**Description**

As per the intended behavior of the contract, there is a maximum supply limit for the token. Whenever the owner tries to mint a token, there is a check in place which ensures that the total supply after the mint does not exceed. However, in the constructor of the contract, there is a mint function that mints 100000000 tokens to the owner of the contract. Since the value is hard coded this isn't a major issue but checks should be added to make sure that total supply is not more than maximum supply limit.

**Remediation**

Make sure that total tokens to mint value in constructor is not changed and a check is added to make sure total supply can never be more than maximum supply.

**Status**

**Fixed**

# Informational Issues

## A.2 Unlocked pragma (pragma solidity ^0.8.9)

**Description**

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

**Recommendation**

It is recommended to avoid floating pragma compilers and ensure to lock them. This particularly ensures that the latest update doesn't expose this contract to hack due to new features and bugs that could come with the newer version.

**Status**

**Fixed**

## A.3 Missing Zero Address check

```
  constructor(address _trustedForwarder) ERC2771Context(_trustedForwarder)
ERC20("GazeCoin Metaverse Token", "GZE") {
      _mint(msg.sender, 100000000 ether);
      _owner = msg.sender;
  }
```

**Description**

Contracts lack zero address checks, hence are prone to be initialized with zero addresses.

**Remediation**

Consider adding zero address checks in order to avoid risks.

**Status**

**Fixed**

## A.4 Absence of Comments

**Description**

Contracts should have appropriate comments to help users and developers understand the underlying motives behind every function in the contract. This creates a form of confidence for anyone interacting with the contract.

**Recommendation**

Here, the Natspec format is recommended to help outline the motive. This model of comments is detailed and quick to understand when present in comments.

**Status**

**Fixed**

# Functional Testing

**Some of the tests performed are mentioned below**

- Should get the name of the token.
- Should get symbol of the token
- Should get the decimals of the token.
- Should get the total supply of the token
- Should get the trustForwarder of the contract.
- Should revert if not a trust forwarder.
- Should get the balance of the deployer.
- Should get the new total supply
- Should fail when minting exceeds 2500000000 ethers
- Should mint, burn and reduce the total supply of the token.

# Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

# Closing Summary

In this report, we have considered the security of GazeCoin. We performed our audit according to the procedure described above.

Some issues of Low and informational severity were found, some suggestions and best practices are also provided in order to improve the code quality and security posture.

In the End, GazeCoin Team Resolved all Issues.

# Disclaimer

QuillAudits smart contract audit is not a security warranty, investment advice, or an endorsement of the GazeCoin Platform. This audit does not provide a security or correctness guarantee of the audited smart contracts.

The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the GazeCoin Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

# About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

**500+**
Audits Completed

**$15B**
Secured

**500K**
Lines of Code Audited

## Follow Our Journey

# Audit Report
# June, 2022

For

GazeCoin

QuillAudits


Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com