Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

# SMART CONTRACT

## Security Audit Report

Project: Metaverse HQ
Platform: Ethereum
Language: Solidity
Date: January 10th, 2025

# Table of contents

THIS IS SECURITY AUDIT REPORT DOCUMENT AND WHICH MAY CONTAIN INFORMATION WHICH IS CONFIDENTIAL. WHICH INCLUDES ANY POTENTIAL VULNERABILITIES AND MALICIOUS CODES WHICH CAN BE USED TO EXPLOIT THE SOFTWARE. THIS MUST BE REFERRED INTERNALLY AND ONLY SHOULD BE MADE AVAILABLE TO THE PUBLIC AFTER ISSUES ARE RESOLVED.


**Email: audit@EtherAuthority.io**

# Introduction

EtherAuthority was contracted by Metaverse HQ to perform the Security audit of the HQ Token smart contract code. The audit was performed using manual analysis and automated software tools. This report presents all the findings regarding the audit performed on January 10th, 2025.

**The purpose of this audit was to address the following:**

- Ensure that all claimed functions exist and function correctly.
- Identify any security vulnerabilities that may be present in the smart contract.

# Project Background

This Solidity code defines a basic ERC20 token called Metaverse_HQ. Here's a summary of its components:

1. **IERC20 Interface**: The contract implements the standard ERC20 interface for compatibility with other smart contracts and dApps.
2. **IERC20Metadata Interface**: Provides additional metadata functions such as name, symbol, and decimals.
3. **Context Contract**: Utility for retrieving the msg.sender and msg.data.
4. **ERC20 Contract**: Implements the core ERC20 functionality:
    - Token transfers (transfer and transferFrom)
    - Allowances (approve, increaseAllowance, decreaseAllowance)
    - Events for Transfer and Approval
5. **Metaverse_HQ Token Contract**:
    - Inherits from the ERC20 base contract.
    - The constructor initializes the token with a name, symbol, and total supply.
    - All tokens are minted to the deployer's address during contract deployment.

# Audit scope

| Name | Code Review and Security Analysis Report for Metaverse HQ Smart Contract |
|---|---|
| Platform | Ethereum / Solidity |
| File | Metaverse_HQ.sol |
| Smart Contract | 0xde6AcEAF7F2dCEB3d425643C5F85351f2B38FcdE |
| Audit Date | January 10th, 2025 |

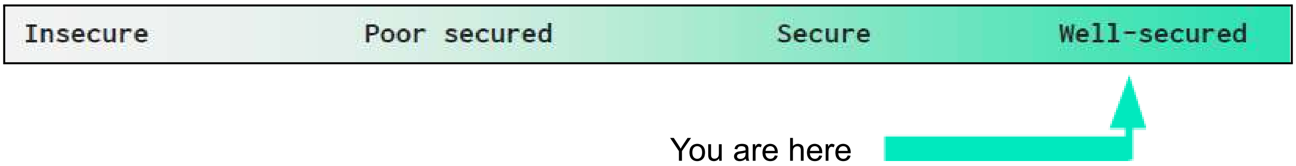
# Claimed Smart Contract Features

| Claimed Feature Detail | Our Observation |
|---|---|
| **Token Details:**<br>• Name: Metaverse HQ<br>• Symbol: HQ<br>• Decimals: 18<br>• Total Supply: 1 billion | **YES, This is valid.** |
| **Key Features of Metaverse_HQ Token:**<br>1. **ERC-20 Standard Compliance:**<br>○ Fully compliant with the ERC-20 standard for fungible tokens.<br>○ Compatible with decentralized exchanges (DEXs) and wallets like MetaMask.<br>2. **Fixed Supply:**<br>○ The supply is fixed at deployment time, and controlled by the constructor.<br>○ No additional minting after deployment, ensuring a deflationary model of tokens are burned.<br>3. **Token Ownership:**<br>○ Full supply is allocated to the deployer's wallet at launch.<br>○ The deployer can distribute tokens manually or send them to a liquidity pool.<br>4. **No Built-in Tax Mechanism:**<br>○ No buy/sell taxes or reflection mechanisms.<br>○ Transactions are free of extra fees or deductions.<br>5. **No Pausing or Governance:**<br>○ The contract does not have admin controls | **YES, This is valid.** |

| | |
|---|---|
| for pausing or modifying token behavior.<br>○ No governance mechanism is included. | |
| **Other Specification:**<br>● This contract does not have any ownership control, hence it is 100% decentralized. | **YES, This is valid.** |

# Audit Summary

According to the standard audit assessment, Customer`s solidity-based smart contracts are **"Well Secured"**. This contract does not have any ownership control, hence it is 100% decentralized.




We used various tools like Slither, Solhint, and Remix IDE. At the same time, this finding is based on a critical analysis of the manual audit.

All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Audit Overview section. The general overview is presented in the AS-IS section and all identified issues can be found in the Audit overview section.

**We found 0 critical, 0 high, 0 medium, 0 low, and 0 very low-level issues.**

**Investor Advice:** A technical audit of the smart contract does not guarantee the ethical nature of the project. Any owner-controlled functions should be executed by the owner with responsibility. All investors/users are advised to do their due diligence before investing in the project.

## Technical Quick Stats

| Main Category | Subcategory | Result |
|---|---|---|
| Contract Programming | The solidity version is not specified | Passed |
| | The solidity version is too old | Passed |
| | Integer overflow/underflow | Passed |
| | Function input parameters lack check | Passed |
| | Function input parameters check bypass | Passed |
| | Function access control lacks management | Passed |
| | Critical operation lacks event log | Passed |
| | Human/contract checks bypass | Passed |
| | Random number generation/use vulnerability | N/A |
| | Fallback function misuse | Passed |
| | Race condition | Passed |
| | Logical vulnerability | Passed |
| | Features claimed | Passed |
| | Other programming issues | Passed |
| Code Specification | Function visibility not explicitly declared | Passed |
| | Var. storage location not explicitly declared | Passed |
| | Use keywords/functions to be deprecated | Passed |
| | Unused code | Passed |
| Gas Optimization | "Out of Gas" Issue | Passed |
| | High consumption 'for/while' loop | Passed |
| | High consumption 'storage' storage | Passed |
| | Assert() misuse | Passed |
| Business Risk | The maximum limit for mintage is not set | Passed |
| | "Short Address" Attack | Passed |
| | "Double Spend" Attack | Passed |

**Overall Audit Result: PASSED**

## Business Risk Analysis

| Category | Result |
|---|---|
| Buy Tax | 0% |
| Sell Tax | 0% |
| Cannot Buy | No |
| Cannot Sell | No |
| Max Tax | 0% |
| Modify Tax | No |
| Fee Check | No |
| Is Honeypot | Not Detected |
| Trading Cooldown | Not Detected |
| Can Pause Trade? | No |
| Pause Transfer? | Not Detected |
| Max Transaction amount? | No |
| Is it Anti-whale? | Not Detected |
| Is Anti-bot? | Not Detected |
| Is it a Blacklist? | Not Detected |
| Blacklist Check | No |
| Can Mint? | No |
| Is it a Proxy? | No |
| Can Take Ownership? | No |
| Hidden Owner? | Not Detected |
| Self Destruction? | Not Detected |
| Auditor Confidence | High |

**Overall Audit Result: PASSED**

## Code Quality

This audit scope has 1 smart contract. Smart contracts contain Libraries, Smart contracts, inherits, and Interfaces. This is a compact and well-written smart contract.

The libraries in the HQ Token are part of its logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned a specific address and its properties/methods can be reused many times by other contracts in the HQ Token.

The Metaverse HQ team has not provided scenario and unit test scripts, which would help to automatically determine the integrity of the code.

The smart contracts comment on code parts not well commented, using Ethereum's NatSpec commenting style, which is good.

## Documentation

We were given an HQ Token smart contract code in the form of an etherscan.io weblink.

As mentioned above, the code parts are not well commented on. but the logic is straightforward. So, it is easy to understand the programming flow and complex code logic quickly. Comments are constructive in understanding the overall architecture of the protocol.

## Use of Dependencies

As per our observation, the libraries used in this smart contract infrastructure are based on well-known industry standard open-source projects.

Apart from libraries, its functions are not used in external smart contract calls.

# AS-IS overview

### Functions

| Sl. | Functions | Type | Observation | Conclusion |
|---|---|---|---|---|
| 1 | constructor | write | Passed | No Issue |
| 2 | name | read | Passed | No Issue |
| 3 | symbol | read | Passed | No Issue |
| 4 | decimals | read | Passed | No Issue |
| 5 | totalSupply | read | Passed | No Issue |
| 6 | balanceOf | read | Passed | No Issue |
| 7 | transfer | write | Passed | No Issue |
| 8 | allowance | read | Passed | No Issue |
| 9 | approve | write | Passed | No Issue |
| 10 | transferFrom | write | Passed | No Issue |
| 11 | decreaseAllowance | write | Passed | No Issue |
| 12 | increaseAllowance | write | Passed | No Issue |
| 13 | _transfer | internal | Passed | No Issue |
| 14 | _mint | internal | Passed | No Issue |
| 15 | _burn | internal | Passed | No Issue |
| 16 | _approve | internal | Passed | No Issue |
| 17 | _spendAllowance | internal | Passed | No Issue |
| 18 | _beforeTokenTransfer | internal | Passed | No Issue |
| 19 | _afterTokenTransfer | internal | Passed | No Issue |

## Severity Definitions

| Risk Level | Description |
|---|---|
| Critical | Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc. |
| High | High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g. public access to crucial |
| Medium | Medium-level vulnerabilities are important to fix; however, they can't lead to tokens lose |
| Low | Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets, that can't have a significant impact on execution |
| Lowest / Code Style / Best Practice | Lowest-level vulnerabilities, code style violations, and info statements can't affect smart contract execution and can be ignored. |

# Audit Findings

## Critical Severity

No critical severity vulnerabilities were found.

## High Severity

No high-severity vulnerabilities were found.

### Medium

No medium-severity vulnerabilities were found.

### Low

No low-severity vulnerabilities were found.

### Very Low / Informational / Best practices:

No very low-severity vulnerabilities were found.


**Email: audit@EtherAuthority.io**

## Centralization Risk

The HQ Token smart contract does not have any ownership control, **hence it is 100% decentralized.**

Therefore, there is **no** centralization risk.


**Email: audit@EtherAuthority.io**

# Conclusion

We were given a contract code as an etherscan.io weblink, and we used all possible tests based on the given objects. We have not observed any issues. **So, the smart contract is ready for mainnet deployment.**

Since possible test cases can be unlimited for such smart contracts protocol, we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover the maximum possible test cases to scan everything.

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. Smart Contract's high-level description of functionality was presented in the As-is overview section of the report.

The audit report contains all security vulnerabilities and other issues found in the reviewed code.

The security state of the reviewed smart contract, based on standard audit procedure scope, is **"Well Secured".**

# Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of the systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

**Manual Code Review:**

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

**Vulnerability Analysis:**

Our audit techniques included manual code analysis, user interface interaction, and white box penetration testing. We look at the project's website to get a high-level understanding of the functionality of the software under review. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

**Documenting Results:**

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, and then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this, we analyze the feasibility of an attack in a live system.

**Suggested Solutions:**

We search for immediate mitigations that live deployments can take, and finally, we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

# Disclaimers

## EtherAuthority.io Disclaimer

EtherAuthority team has analyzed this smart contract in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

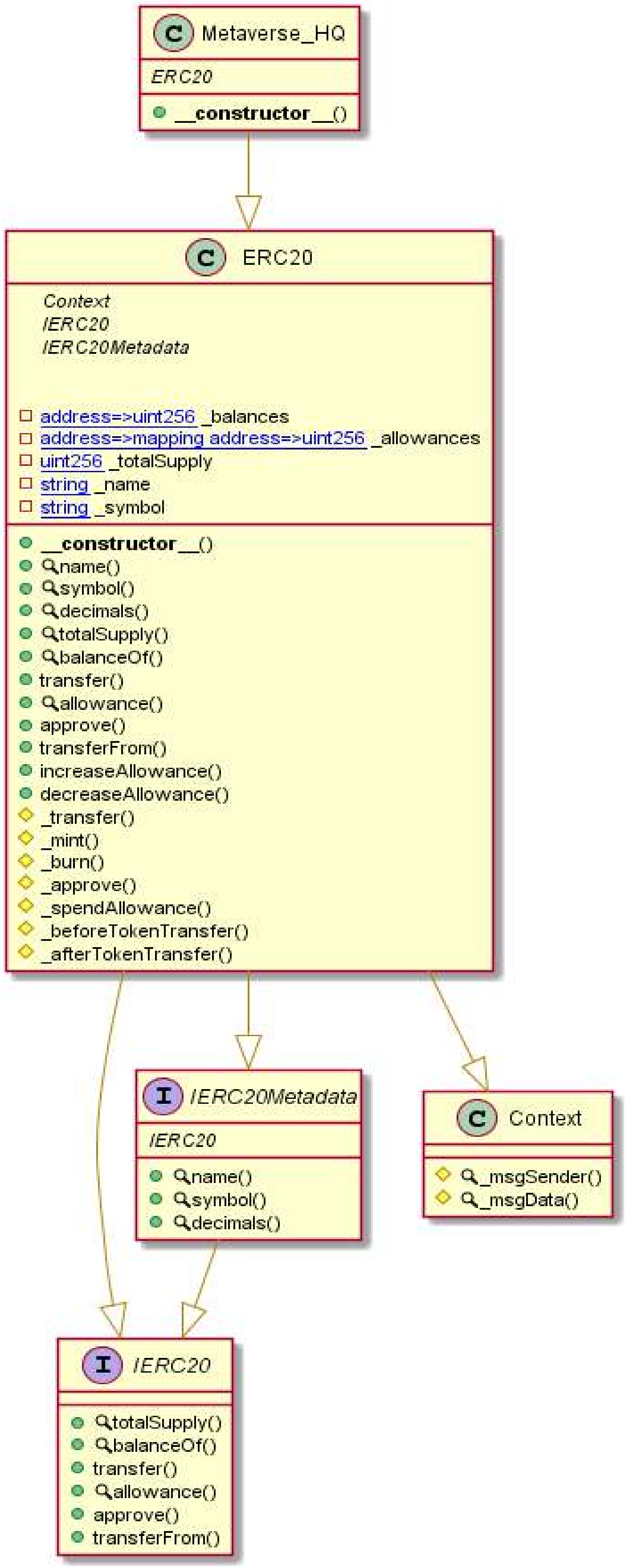
Due to the fact that the total number of test cases is unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While we have done our best to conduct the analysis and produce this report, it is important to note that you should not rely on this report only. We also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

## Technical Disclaimer

Smart contracts are deployed and executed on the blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can't guarantee the explicit security of the audited smart contracts.

# Appendix

## Code Flow Diagram - Metaverse HQ



# Slither Results Log

## Slither Log >> Metaverse_HQ.sol

```
INFO:Detectors:
Metaverse_HQ.constructor(string,string,uint256).name (Metaverse_HQ.sol#446) shadows:
	- ERC20.name() (Metaverse_HQ.sol#138-140) (function)
	- IERC20Metadata.name() (Metaverse_HQ.sol#90) (function)
Metaverse_HQ.constructor(string,string,uint256).symbol (Metaverse_HQ.sol#447) shadows:
	- ERC20.symbol() (Metaverse_HQ.sol#146-148) (function)
	- IERC20Metadata.symbol() (Metaverse_HQ.sol#95) (function)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
Context._msgData() (Metaverse_HQ.sol#109-111) is never used and should be removed
ERC20._burn(address,uint256) (Metaverse_HQ.sol#353-369) is never used and should be
removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Version constraint ^0.8.0 contains known severe issues
(https://solidity.readthedocs.io/en/latest/bugs.html)
	- FullInlinerNonExpressionSplitArgumentEvaluationOrder
	- MissingSideEffectsOnSelectorAccess
	- AbiReencodingHeadOverflowWithStaticArrayCleanup
	- DirtyBytesArrayToStorage
	- DataLocationChangeInInternalOverride
	- NestedCalldataArrayAbiReencodingSizeValidation
	- SignedImmutables
	- ABIDecodeTwoDimensionalArrayMemory
	- KeccakCaching.
It is used by:
	- ^0.8.0 (Metaverse_HQ.sol#4)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Contract Metaverse_HQ (Metaverse_HQ.sol#443-454) is not in CapWords
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-c
onventions
INFO:Slither:Metaverse_HQ.sol analyzed (5 contracts with 93 detectors), 6 result(s) found
```


**Email: audit@EtherAuthority.io**

# Solidity Static Analysis

## Metaverse_HQ.sol

Gas costs:
Gas requirement of function Metaverse_HQ.decreaseAllowance is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 273:4:

Constant/View/Pure functions:
ERC20._beforeTokenTransfer(address,address,uint256) : Potentially should be constant/view/pure but is not.
Pos: 424:4:

Similar variable names:
Metaverse_HQ.(string,string,uint256) : Variables have very similar names "_totalSupply" and "totalSupply_".
Pos: 450:26:

Guard conditions:
Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
Pos: 354:8:


**Email: audit@EtherAuthority.io**

## Solhint Linter

**Metaverse_HQ.sol**

```
Compiler version ^0.8.0 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:3
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:129
Error message for require is too long
Pos: 9:275
Error message for require is too long
Pos: 9:298
Error message for require is too long
Pos: 9:299
Error message for require is too long
Pos: 9:304
Error message for require is too long
Pos: 9:353
Error message for require is too long
Pos: 9:358
Error message for require is too long
Pos: 9:384
Error message for require is too long
Pos: 9:385
Code contains empty blocks
Pos: 94:423
Code contains empty blocks
Pos: 93:439
Contract name must be in CamelCase
Pos: 1:442
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:444
```

**Software analysis result:**

This software reported many false positive results and some are informational issues. So, those issues can be safely ignored.


**Email: audit@EtherAuthority.io**

EtherAuthority



**Email: audit@EtherAuthority.io**