QuillAudits

Audit Report
March, 2022

For

METAGAMEHUB

Contents

Overview

MiniMe token

The MiniMe token is a cloneable ERC 20 token. It is designed to easily spawn tokens that have the same balance distribution as the parent token at any given block number.

Scope of the Audit

The scope of this audit was to analyze **MineMe Token smart contract** for quality, security, and correctness.

MiniMe Token Contract: MiniMeToken | 0x8765b1a0eb57ca49be7eacd35b24a574d0203656 (etherscan.io)

Checked Vulnerabilities

We have scanned the smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered:

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- Exception Disorder
- Gasless Send
- Use of tx.origin
- Malicious libraries
- Compiler version not fixed
- Address hardcoded
- Divide before multiplying
- Integer overflow/underflow
- ERC20 transfer() does not return boolean
- ERC20 approve() race
- Dangerous strict equalities
- Tautology or contradiction
- Return values of low-level calls
- Missing Zero Address Validation
- Private modifier
- Revert/require functions
- Using block.timestamp
- Multiple Sends
- Using SHA3
- Using suicide
- Using throw
- Using inline assembly

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
Slither, MythX, Truffle, Remix, Ganache, Solidity Metrics

Issue Categories

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below.

Risk-level	Description
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.

Number of issues per severity

Type	High	Medium	Low	Informational
Open	0	0	0	0
Acknowledged	0	3	2	3
Closed	0	0	0	0

Issues Found – Code Review / Manual Testing

High severity issues

No issues found

Medium severity issues

1. **Incorrect version of solidity**

   Contracts are using old version of solidity and it contains use of deprecated code/keywords, solc frequently releases new compiler versions. Using an old version prevents access to new Solidity security checks.

   Recommendation
   We recommend the use of the latest solc version.

   Status: Acknowledged

2. **transferFrom(),transfer() returns false bool instead of reverting on failure.**

   [Line 202] transferFrom() returns ‘false’ if allowance is less than amount **,[line 226]** doTransfer() function used by transferFrom() and transfer() returns ‘false’ on failure If the amount being transferred is more than the balance of the account instead of reverting.

   Recommendation
   Use require to revert the transaction instead returning bool false.
   If using the version which returns false bool then It is safer to wrap such calls into require() statements to these failures when calling from caller(can be controller contract in this case).

   Reference: ForceDAO Hack

   Status: Acknowledged

3. Centralization risk of Minimetoken

[Line 202] transferFrom() Transfer of MiniMeTokens can be disabled by the controller of the MiniMeToken. Thus, even if the deposit period of Staking Pools have started, the users will not be able to deposit their tokens.

Also if the pool is Terminated, and the transfer of minimetokens is disabled, the users will not be able to withdraw any of their tokens, leading to a potential Denial of Service attack.

Recommendation
Make sure that the controller of the MiniMeToken is trustworthy.

Reference: ForceDAO Hack

Status: Acknowledged

Low severity issues

4. Critical Address change

[Line 70] changeController function changes controller's address which has many privileges . If by mistake current controller address passes function argument which is wrong address that current controller dont want to set then current controller can lose all privileges, In this case controller needs a mechanism from which current controller can first set and then confirm the address for which he granted the admin/ controller privileges.

Recommendation
Changing critical addresses in contracts should be a two-step process where the first transaction (from the old/current address) registers the new address (i.e. grants ownership) and the second transaction (from the new address) replaces the old address with the new one (i.e. claims ownership). This gives an opportunity to recover from incorrect addresses mistakenly used in the first step. If not, contract functionality might become inaccessible.

Status: Acknowledged

5. **Usage of isContract() does not guarantee that an address is a Contract or not**

   [Line 70] Line: 223, 229, 268, 502
   isContract() function has been used to determine whether the address is a contract or not. It is unsafe to assume that an address for which this function returns false is an externally-owned account (EOA) and not a contract. Among others, isContract will return false for the following types of addresses:
   a) an externally-owned account
   b) a contract in construction
   c) an address where a contract will be created
   d) an address where a contract lived, but was destroyed

   Recommendation
   Avoid using isContract unless it is absolutely sure as to the effects and impacts of when the isContract function fails

   Status: Acknowledged

Informational issues

6. **Public function that could be declared external**

   [line 70] changeController function is declared “public”, public functions that are never called by the contract should be declared external to save gas.

   Recommendation
   Use the external attribute for functions never called from the contract to save gas fees.

   Status: Acknowledged

7. **Using inline assembly**

   [Line 229] isContract function has an inline assembly, after analysis it seems it's not impacting contract security, but assembly bypasses several important safety features and checks of Solidity,The use of assembly is error-prone and should be avoided.

Recommendation
Avoid the use of evm assembly.

Status: Acknowledged

8. Usage of Deprecated keywords

a) Line: 66, 152 Defining constructor with the same name as contract is deprecated and it is advised to use the name constructor instead

b) Line: 224, 238 Using of var keyword is deprecated and its usage should be avoided.

c) Line: 242, 274, 402, 421, 437 and 545 Invoking events without emit keyword is deprecated and should be avoided.

d) Line: 538, Using balance inherited from the address type is deprecated. It is advised to convert the contract to address

type to access the member, for example use "address(contract).balance" instead.

Recommendation
Avoid the usage of deprecated keywords

Status: Acknowledged

Functional Testing Results

Some of the tests performed are mentioned below:

Contract: Controlled

☑ Should be able to change controller address
☑ Should revert if address without controller privileges tries to change controller

Contract: MiniMeToken

☑ Should be able to generate tokens
☑ Should be able to destroy tokens
☑ Should be able to transfer tokens
☑ Should be able to approve tokens
☑ Should be able to transferFrom tokens
☑ Should revert if transfer functionality is not enabled
☑ Should revert on zero address
✗ Should revert on transfer failure
✗ Should revert on transferFrom failure

Closing Summary

Some issues of Medium and Low severity were found, which has been Acknowledged by the Auditee. Some suggestions and best practices are also provided in order to improve the code quality and security posture.

Disclaimer

QuillAudits smart contract audit is not a security warranty, investment advice, or an endorsement of the MiniMe Token.

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 MiniMe Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

Audit Report
March, 2022

For

METAGAMEHUB

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com