QuillAudits

Audit Report
February, 2022

For

Contents

Overview

The Dartter 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.

Dartter Token Contract: https://bscscan.com/address/0xC9F171D39aB17b63954b9d78b304Cf122D68356#code

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
- Exception Disorder
- Gasless Send
- Use of tx.origin
- Malicious libraries
- Compiler version not fixed
- Address hardcoded
- Divide before multiply
- Integer overflow/underflow
- BEP20 transfer() does not return boolean
- BEP20 approve() race condition
- 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
- BEP20 Standard

Techniques and Methods

Throughout the audit of the 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/BEP20 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
Mythril, Slither, SmartCheck, Surya, Solhint.

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	0	0	4
Closed	0	0	0	0

Functional Testing Results

Some of the tests performed are mentioned below:

- Should be able to transfer tokens PASS
- Should be able to approve tokens PASS
- Should be able to transfer ownership PASS
- Should be able to burn tokens PASS
- Should be able to increase and decrease allowance PASS
- Should revert if recipient is zero address PASS
- Should revert if caller is not owner PASS
- Should revert if account is already excluded PASS
- Should revert if the transfer amount exceeds the balance PASS
- Tax should not be removed only when both sender and recipient are excluded PASS

Issues Found

High severity issues

No issues found

Medium severity issues

No issues found

Low severity issues

No issues found

Informational issues

- **Centalization risk**

  Some functions that use the “onlyOwner” pose a centralization risk to the contracts the “onlyOwner” modifier may hinder with the working of smart contract in case of loss of the public key. Effected functions are -
  - setBPAddrss()
  - setBpEnabled()
  - setBotProtectionDisableForever()
  - updateTax()
  - setTaxEnabled()
  - setTaxStoreEnabled()
  - excludeAccount()
  - includeAccount()
  - burn()

Recommendation
Instead of using a one-step process for assigning/changing these values, switching to a two-step process should be considered.

Status: Acknowledged

- **Comparing with boolean constant**

  Function setBPAddrss compare with a boolean constant, whereas Boolean constants can be used directly and do not need to be compared to true or false

```
function setBotProtectionDisableForever() external onlyOwner {
    require(BPDisabledForever == false);
    BPDisabledForever = true;
}
```

  Recommendation
  Boolean constants can be used directly and do not need to be compared to true or false.

  Status: Acknowledged

- **Race Condition**

  _approve function contains race condition and there's a possibility to frontun transaction call for this function. resulting approval of tokens more than the sender wanted to send if the receiver tries to frontrun.

  Recommendation
  There should be a check statement which will check if current allowance is 0 before setting it to another value for the same spender.

  Status: Acknowledged

- **Floating Pragma**

  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.

```
pragma solidity ^0.6.12;

// SPDX-License-Identifier: Unlicensed
interface IBEP20 {
    function totalSupply() external view returns (uint256);

    function decimals() external view returns (uint8);

    function symbol() external view returns (string memory);
```

Recommendation
Lock the pragma version for the compiler version that is chosen.

Status: Acknowledged

Closing Summary

Some issues of Informational severity were found, which has been Acknowledged by the Dartter Team.

Disclaimer

Quillhash audit is not a security warranty, investment advice, or an endorsement of the Dartter Token. 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 Dartter team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

Audit Report
February, 2022

For

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com