QuillAudits

Audit Report November, 2023

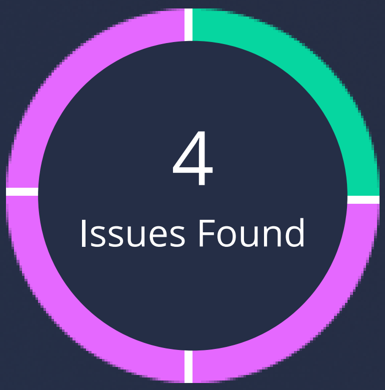
For

Table of Content

Executive Summary

Project Name	Decats
Project URL	*https://decats.io/*
Overview	$DECATS is a community memecoin created out of the community of DeBank, a fast growing and very exciting new SocialFi network.
Timeline	16th November 2023 - 20th November 2023
Audit Scope	The scope of this audit was to analyze the Decats codebase for quality, security, and correctness.
Source Code	*https://polygonscan.com/address/0x198f1d316aad1c0bfd36a79bd1a8e9dba92daa18#code*
Method	Manual Review, Functional Testing, Automated Testing, etc. All the raised flags were manually reviewed and re-tested to identify any false positives.
Updated Code Received	NA
Second Review	NA
Fixed In	NA

Number of Security Issues per Severity



🟥 High　🟨 Medium

🟩 Low　🟪 Informational

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

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 level

Techniques and Methods

Throughout the audit of smart contracts, 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 analyzed 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

A 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 analyzed, 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 behavior 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

Hardhat, Foundry.

Types of Severity

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.

High Severity Issues

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 Severity Issues

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 Severity Issues

Low-level severity issues can cause minor impact and 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 four 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.

Low Severity Issues

1. Not protected for the ERC20 approval race condition

Path

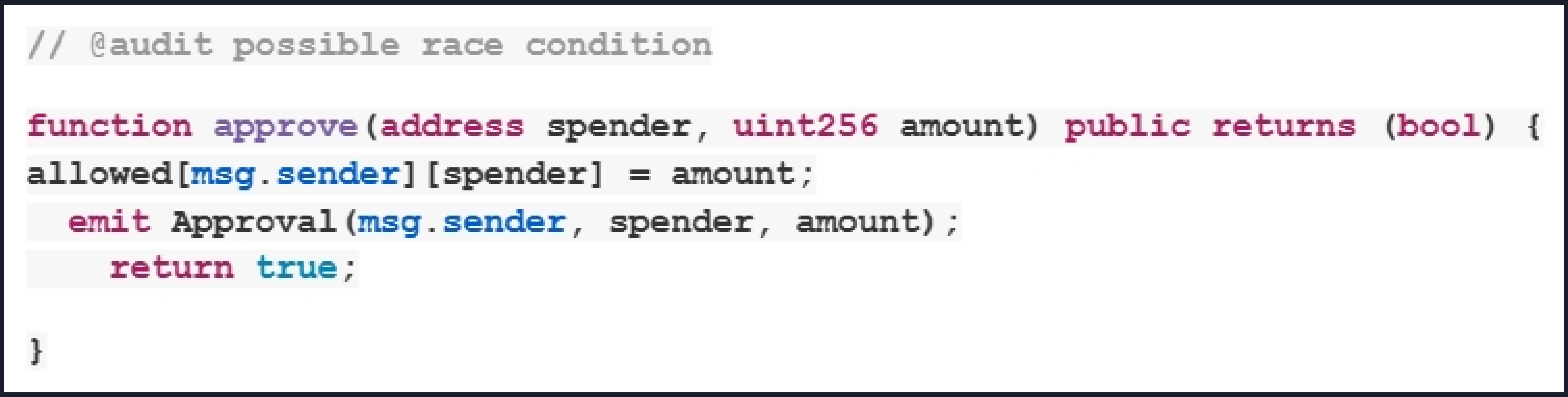
InitializableERC20.sol

Function

Approve

Description

There is no protection for race condition through the approve function.



```
// @audit possible race condition

function approve(address spender, uint256 amount) public returns (bool) {
allowed[msg.sender][spender] = amount;
  emit Approval(msg.sender, spender, amount);
    return true;

}
```

For Example, if Alice approves 10 tokens to Bob, later wants to change the approval to 5 tokens and calls the approve function which will create a transaction. Bob can look into the mempool and notice that approval is changed. Then, Bob will send a transaction (call transferfrom) with a higher gas price than the approval transaction to transfer 10 tokens to himself. Later, 5 tokens approval transaction is executed, and Bob can get 5 more tokens as well. Alice wanted to send 5 tokens but instead gave out 15 tokens.

The contract is not inheriting OpenZeppelin's ERC20 contract.

Recommendation

Consider adding a check to confirm that approval is zero or include **increaseAllowance** and **decearseAllowance** functionalities in the contract.

Status

Acknowledged

Informational Issues

2. Front running of **init** function

Path

InitializableERC20.sol

Description

init function basically acts as an initialise function for the contract. creator, totalSupply, name, decimals, and symbol are set through this function. As well as, the creator address's balance is set with total supply.

Attackers/Searchers can front-run the init call transaction by looking at the mempool. Through this, they can set the values mentioned above and steal tokens. This is happening because the function is public, and there is no check on who the caller is.

This is not a high-severity issue because the contract is deployed and live already.

Recommendation

Consider including the owner of the contract through the constructor to avoid front running of **init** function if planned to deploy the contract in different chain.

Status

Acknowledged

3. Reduce number storage slots used

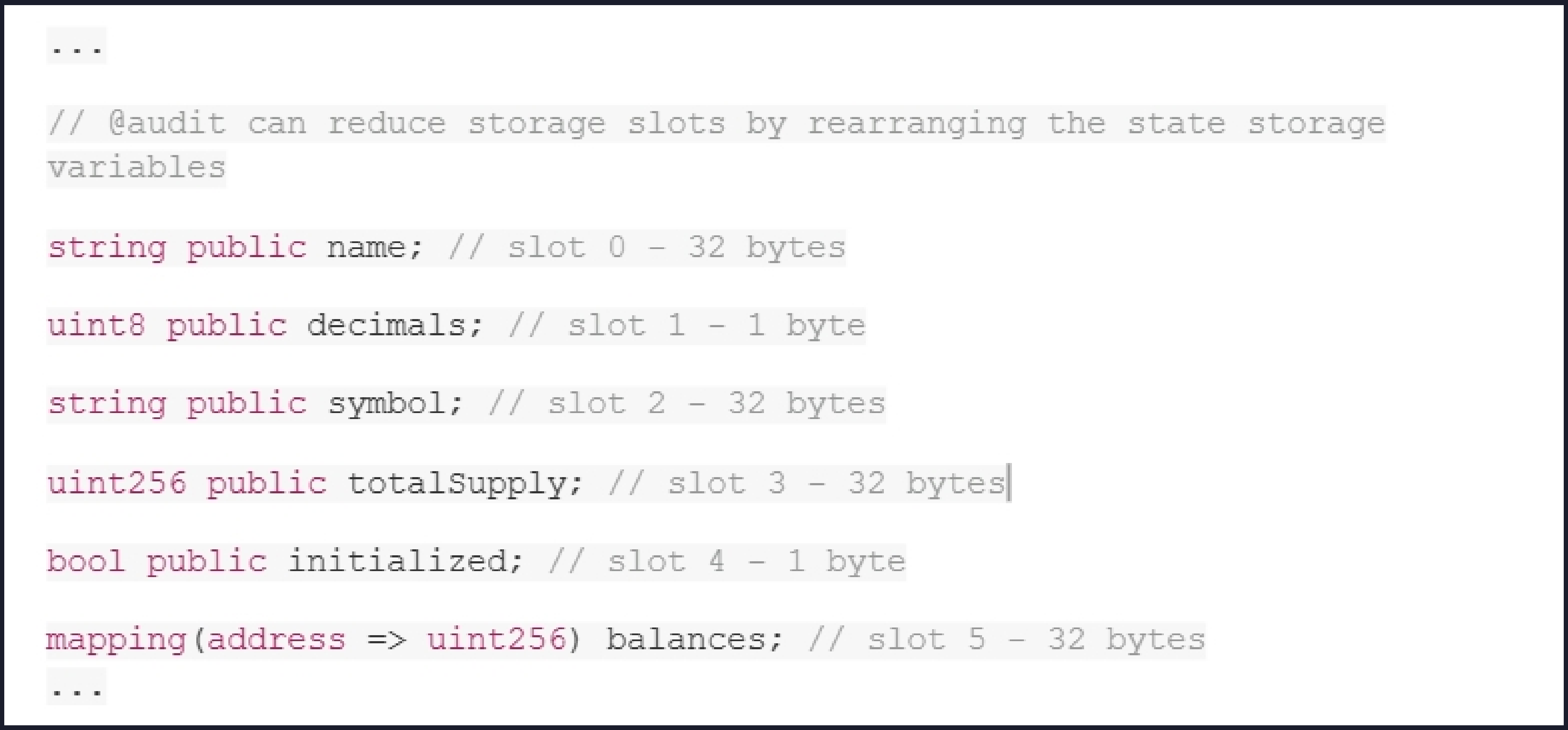
Path

InitializableERC20.sol

Description

Each slot in the storage is 32 bytes in size. **decimals** and **initialized** only use 1 byte, but both are defined between 32-byte size variables. This creates new slots just for 1-byte variables, and total number of slots is 6.

3. Reduce number storage slots used



```
...

// @audit can reduce storage slots by rearranging the state storage
variables

string public name; // slot 0 - 32 bytes

uint8 public decimals; // slot 1 - 1 byte

string public symbol; // slot 2 - 32 bytes

uint256 public totalSupply; // slot 3 - 32 bytes

bool public initialized; // slot 4 - 1 byte

mapping(address => uint256) balances; // slot 5 - 32 bytes
...
```

Order of variables can be **balances**, **totalSupply**, **name**, **symbol**, **decimals**, and **initialized**. This will reduce the total number of slots to 5.

Recommendation

Consider changing the order of state variables in the contract to reduce the usage of storage.

Status

Acknowledged

4. Use the latest version of Solidity

Path

InitializableERC20.sol

Description

The contract is using 0.6.9 solidity version, which is old and contains vulnerabilities. Using the latest stable solidity version will prevent vulnerabilities.

Recommendation

Consider using the latest version of Solidity in the contract.

Status

Acknowledged

Functional Tests

Some of the tests performed are mentioned below:

- ✓ Should allow transfer of tokens from one address to another address
- ✓ Should allow users to approve tokens for other addresses
- ✓ Should allow users to transfer tokens on behalf of another address when approved
- ✗ Should allow users to increase and decrease the allowances of tokens

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 the Decats codebase. 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.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in Decats smart contracts. However, it is important to understand that no security audit can guarantee complete protection against all possible security threats. QuillAudits audit reports are based on the information provided to us at the time of the audit, and we cannot guarantee the accuracy or completeness of this information. Additionally, the security landscape is constantly evolving, and new security threats may emerge after the audit has been completed.

Therefore, it is recommended that multiple audits and bug bounty programs be conducted to ensure the ongoing security of Decats smart contracts. One audit is not enough to guarantee complete protection against all possible security threats. It is important to implement proper risk management strategies and stay vigilant in monitoring your smart contracts for potential security risks.

QuillAudits cannot be held liable for any security breaches or losses that may occur subsequent to and despite using our audit services. It is the responsibility of the Decats to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

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.

850+
Audits Completed

$30B
Secured

$30B
Lines of Code Audited

Follow Our Journey

Audit Report
November, 2023

For

DECATS

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com