QuillAudits

Audit Report
March, 2021

BeeDAO

Contents

Introduction

This Audit Report mainly focuses on the overall security of BDAO token from BeeDAO. With this report, we have tried to ensure the reliability and correctness of their smart contract by a complete and rigorous assessment of their system's architecture and the smart contract codebase.

Auditing Approach and Methodologies applied

The QuillHash team has performed rigorous testing of the project starting with analyzing the code design patterns in which we reviewed the smart contract architecture to ensure it is well structured and safe use of third-party smart contracts and libraries.

Our team then performed a formal line-by-line inspection of the Smart Contract to find any potential issues like race conditions, transaction-ordering dependence, timestamp dependence, and denial of service attacks.

In the Unit testing Phase, we coded/conducted custom unit tests written for each function in the contract to verify that each function works as expected.

In Automated Testing, We tested the Smart Contract with our in-house developed tools to identify vulnerabilities and security flaws.

The code was tested in collaboration with our multiple team members and this included -

- Testing the functionality of the Smart Contract to determine proper logic has been followed throughout the whole process.
- Analysing the complexity of the code in-depth and detailed, manual review of the code, line-by-line.
- Deploying the code on testnet using multiple clients to run live tests.
- Analysing failure preparations to check how the Smart Contract performs in case of any bugs and vulnerabilities.
- Checking whether all the libraries used in the code are on the latest version.
- Analysing the security of the on-chain data.

Audit Details

Project Name: BeeDAO
Website/BscScan Code (Testnet):

- **BDAO Token:** 0x31214B3573d70077016ceb554e78709F4107D505

Languages: Solidity (Smart contract)
Platforms and Tools: Remix IDE, Truffle, Truffle Team, Ganache, Solhint, VScode, Contract Library, Slither, SmartCheck

Audit Goals

The focus of the audit was to verify that the Smart Contract System is secure, resilient, and working according to the specifications. The audit activities can be grouped into the following three categories:

Security
Identifying security related issues within each contract and the system of contract.

Sound Architecture
Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.

Code Correctness and Quality
A full review of the contract source code. The primary areas of focus include:

- Accuracy
- Readability
- Sections of code with high complexity

Issue Categories

Every issue in this report was assigned a severity level from the following:

High severity issues

Issues on this level are critical to the smart contract's performance/ functionality and should be fixed before moving to a live environment.

Medium severity issues

Issues on this level could potentially bring problems and should eventually be fixed.

Low severity issues

Issues on this level are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

Number of issues per severity

	High	Medium	Low	Informational
Open	0	0	1	2
Closed	0	0	0	0

Manual Audit

For this section, the code was tested/read line by line by our developers. We also used Remix IDE's JavaScript VM and Kovan networks to test the contract functionality.

High level severity issues

No issues found

Medium level severity issues

No issues found

Low level severity issues

1. Methods should not be included in the contract which are internal methods but never getting called from the contract, to reduce gas cost while deploying the contract.

   Since _burnFrom is never used, it is recommended to remove it before deployment.

```
function _burnFrom(address account, uint256 amount) internal {
    _burn(account, amount);
    _approve(account, _msgSender(),
_allowances[account][_msgSender()].sub(amount, "BEP20: burn
amount exceeds allowance"));
  }
```

Recommendations

1. A function with a public visibility modifier that is not called internally should be set to external visibility to increase code readability. Moreover, in many cases, functions with external visibility modifier spend less gas compared to functions with public visibility modifier. Following functions can be declared external:

   - **increaseAllowance(address,uint256) (BDAO.sol#469-472)**
   - **decreaseAllowance(address,uint256) (BDAO.sol#488-491)**
   - **mint(uint256) (BDAO.sol#501-504)**
   - **renounceOwnership (BeeDAO.sol#320-323)**
   - **transferOwnership (BeeDAO.sol#329-331)**

2. In BDAOTokens, multiple functions do not check for amounts. It is recommended to add a check for 0 amount. OR check amounts at the application end, before interacting with the contracts.

- In _mint() function (Line number - 538)

```
require(_amount != 0, "BDAO::mint: mint value should not be zero");
```

- In _burn() function (Line number - 557)

```
require(value != 0, "BDAO::burn: burn value should not be zero");
```

- In _transfer() function (Line number - 520)

```
require(_value != 0, "BDAO::transfer: transfer value should not be zero");
```

Automated Testing

Solhint Linting Violations

Solhint is an open-source project for linting solidity code, providing both security and style guide validations. It integrates seamlessly into most mainstream IDEs. We used Solhint as a plugin within our VScode for this analysis. Multiple Linting violations were detected by Solhint, it is recommended to use Solhint's npm package to lint the contract.

Slither

Slither, an open-source static analysis framework. This tool provides rich information about Ethereum smart contracts and has critical properties. While Slither is built as a security-oriented static analysis framework, it is also used to enhance the user's understanding of smart contracts, assist in code reviews, and detect missing optimizations.

The concerns slither raises have already been covered in the manual audit section.

```
'npx truffle compile --all' running (use --truffle-version truffle@x.x.x to use specific version)

Compiling your contracts...
===========================
> Compiling ./contracts/BeeDAO.sol
> Artifacts written to /home/ayush/work/projects/audits/centralex-token/build/contracts
> Compiled successfully using:
   - solc: 0.5.16+commit.9c3226ce.Emscripten.clang


Warning: Both truffle-config.js and truffle.js were found. Using truffle-config.js.

INFO:Detectors:
BDAO.allowance(address,address).owner (BeeDAO.sol#423) shadows:
    - Ownable.owner() (BeeDAO.sol#301-303) (function)
BDAO._approve(address,address,uint256).owner (BeeDAO.sol#578) shadows:
    - Ownable.owner() (BeeDAO.sol#301-303) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
Redundant expression "this (BeeDAO.sol#118)" inContext (BeeDAO.sol#108-121)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
BDAO.constructor() (BeeDAO.sol#355-363) uses literals with too many digits:
    - _totalSupply = 3000000 * (10 ** 18) (BeeDAO.sol#359)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
renounceOwnership() should be declared external:
    - Ownable.renounceOwnership() (BeeDAO.sol#320-323)
transferOwnership(address) should be declared external:
    - Ownable.transferOwnership(address) (BeeDAO.sol#329-331)
```

```
increaseAllowance(address,uint256) should be declared external:
    - BDAO.increaseAllowance(address,uint256) (BeeDAO.sol#469-472)
decreaseAllowance(address,uint256) should be declared external:
    - BDAO.decreaseAllowance(address,uint256) (BeeDAO.sol#488-491)
mint(uint256) should be declared external:
    - BDAO.mint(uint256) (BeeDAO.sol#501-504)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:. analyzed (5 contracts with 72 detectors), 9 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Disclaimer

QuillHash audit is not a security warranty, investment advice, or an endorsement of the BDAO Token. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Summary

The use case of the smart contract is simple and the code is relatively small. Altogether, the code is written and demonstrates effective use of abstraction, separation of concerns, and modularity. There is one low severity issue which can mean more gas consumption during deployment, but no security implications. Therefore, contract is ready for a live deployment.

BeeDAO

QuillAudits

Canada, India, Singapore and United Kingdom

audits.quillhash.com

hello@quillhash.com