QuillAudits

Audit Report
April, 2021

BitDiamond

Contents

Audit Details and Target

1. Contract

https://bscscan.com/address/0x669288ada63ed65eb3770f1c9eeb8956dedaaa47

2. Audit Target

- To find the security bugs and issues regarding security, potential risks and critical bugs.
- Check gas optimization and check gas consumption.
- Check function reusability and code optimisation.
- Test the limit for token transfer and check the accuracy in decimals.
- Check the functions and naming conventions.
- Check the code for proper checks before every function call.
- Event trigger checks for security and logs.
- Checks for constant and function visibility and dependencies.
- Validate the standard functions and checks.
- Check the business logic and correct implementation.
- Automated script testing for multiple use cases including transfers, including values and multi transfer check.
- Automated script testing to check the validations and limit tests before every function call.
- Check the use of data type and storage optimisation.
- Calculation checks for different use cases based on the transaction, calculations and reflected values.

Functions list and audit details

1. Read Functions in Contract

- allowance()
- balanceOf()
- decimals()

- name()
- owner()
- symbol()
- reflectionFromToken()
- tokenFromReflection()
- _getValues()
- _getTBasics()
- getTTransferAmount()
- _getRBasics()
- _getRTransferAmount()
- _getRate()
- _getCurrentSupply()
- _getTaxFee()
- _getMaxTxAmount()
- isExcluded()

2. Write Function in Contract

- approve()
- increaseAllowance()
- totalFees()
- deliver()
- decreaseAllowance()
- renounceOwnership()
- transfer()
- transferFrom()
- transferOwnership()
- _transferToExcluded()
- _transferBothExcluded()
- excludeAccount()

Overview of BitDiamond

BitDiamond is a BEP20 contract that follows all standard guidelines of BEP20 contract development rules and guidelines. The contract creates BEP20 tokens that are compatible with all types of wallets and BEP20 supported platforms. These tokens can be used as safe tokens with all security features like other BEP20 tokens.
Some special functions are added to the contract to manage the business logic of the holder and tax-based logic.
Additional Logic of fees is added in the contract that manages the fees for different types of transactions, values are calculated based on the amount that the user is transferring.
Amount deducted will be deducted in tokens based on the percentage logic of the contract.
Safe Math functions are used to ensure the safety of tokens and save from any type of calculator error that can affect the working of the contract and transfer safe value.

Tokenomics

As per the information provided, the tokens generated will be initially transferred to the contract owner and then further division will be done based on the business logic of the application.
Tokens cannot be directly purchased from the smart contract, so there will be an additional or third-party platform that will help users to purchase the tokens.

Scope of Audit

The scope of this audit was to analyse BitDiamond smart contracts codebase for quality, security, and correctness.

Checked Vulnerabilities

The smart contract is scanned and checked for multiple types of possible bugs and issues. This mainly focuses on issues regarding security, attacks, mathematical errors, logical and business logic issues. Here are some of the commonly known vulnerabilities that are considered:

- TimeStamp dependencies.
- Variable and overflow
- Calculations and checks
- SHA values checks
- Vulnerabilities check for use case
- Standard function checks
- Checks for functions and required checks
- Gas optimisations and utilisation
- Check for token values after transfer
- Proper value updates for decimals
- Array checks
- Safemath checks
- Variable visibility and checks
- Error handling and crash issues
- Code length and function failure check
- Check for negative cases
- D-DOS attacks
- Comments and relevance
- Address hardcoded
- Modifiers check
- Library function use check
- Throw and inline assembly functions
- Locking and unlocking (if any)
- Ownable functions and transfer ownership checks
- Array and integer overflow possibility checks
- Revert or Rollback transactions check

Techniques and Methods

- Manual testing for each and every test cases for all functions.
- Running the functions, getting the outputs and verifying manually for multiple test cases.
- Automated script to check the values and functions based on automated test cases written in JS frameworks.
- Checking standard function and check compatibility with multiple wallets and platforms
- Checks with negative and positive test cases.
- Checks for multiple transactions at the same time and checks d-dos attacks.
- Validating multiple addresses for transactions and validating the results in managed excel.
- Get the details of failed and success cases and compare them with the expected output.
- Verifying gas usage and consumption and comparing with other standard token platforms and optimizing the results.
- Validate the transactions before sending and test the possibilities of

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.
SmartCheck.

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 security of smart contracts.

Code Review / Manual Analysis
Manual Analysis or review of code was done to identify new vulnerability 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 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 possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

Remix IDE, Truffle, Truffle Team, Ganache, Solhint, Mythril, Slither, SmartCheck.

Issue Categories

High severity issues

Issues that must be fixed before deployment else they can create major issues.

Medium level severity issues

These issues will not create major issues in working but affect the performance of the smart contract.

Low level severity issues

These issues are more suggestions that should be implemented to refine the code in terms of gas, fees, speed and code accuracy

Informational

- Code is written in a very concise way which can be more informative and secure by more checks and modifier use.
- The use of variables and naming conventions can be minimized
- Gas use is proper and optimized as per the testing on the Ropsten test network and checking of the old transaction from this contract.
- **Suggestion**: There must be some locking function in case of any loss or security issue so that the admin can lock the contract in case of any security issue to the token.

Number of issues per severity

	High	Medium	Low	Total Issues
Open	0	0	0	0
Closed	0	3	6	9

Issues Found – Code Review / Manual Testing

High severity issues

Not Found in from technical and logical aspects.

Medium severity issues

1. On line 560 _approve function must check 0 value in amount to reduce wastage of fees.

   Resolved

2. 0 value check on function deliver() on the line 535.

   Resolved

3. Function tokenFromReflection() sometime taking very long time or timeout for values above 400 addresses in array _excluded[]

   Resolved

```
function tokenFromReflection(uint256 rAmount) public view returns(uint256) {
    require(rAmount <= _rTotal, "Amount must be less than total reflections");
    uint256 currentRate =  _getRate();
    return rAmount.div(currentRate);
}
```

Low level severity issues

1. Unused variable declared in line number 456.
   Suggestion: Either use the variable or remove the variable _tOwned

   Resolved

2. Symbol names can be improved.
   Suggestion: Symbol should be limited to 3-5 letters.

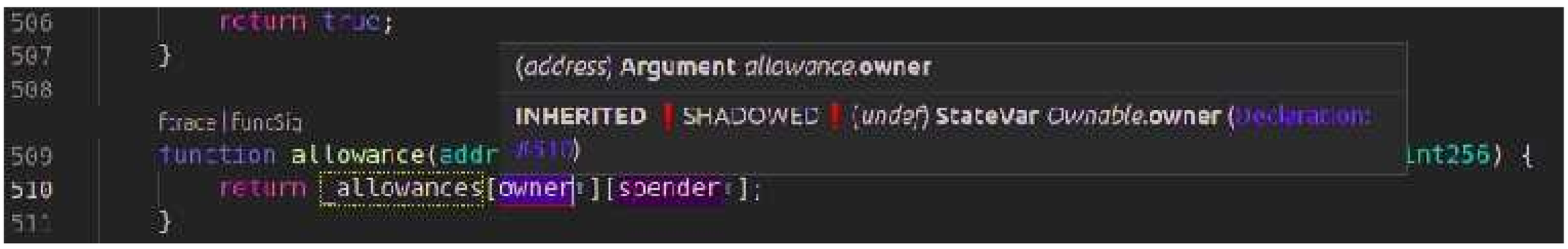
   Resolved

3. Code Complexity check on line 582
   Suggestion: _reflectFee Private functional can be club to reduce the code complexity and gas fees on line 582 with function _transferStandard.

   Resolved

4. Owner variable is shadowed and used multiple times in the contract, suggested to use different variable names to remove discrepancy.

Resolved



5. Uniswap router address can be set as variable and updated with function. In future, if contract updates or version change, we will have to hold on the address.

Resolved

6. Function can be used directly instead of using extra variable

Resolved, as this was only a suggestion for optimization

Closing Summary

Code is written with a good and updated technical approach. This code can be refined and filtered by adding the details and suggestions given above. Changes and dependencies must be checked before implementing the changes.
Some functions have little high gas usage which is needed for designing the business logic of the contract. This code can be optimized by reducing extra used variables and datatypes. Business logic is working fine as manually checked on lesser values or checked with human transactions only. To avoid the logical error code must be checked with all types of values before the deployment.

Disclaimer

QuillHash audit is not a security warranty, investment advice, or an endorsement of the BitDiamond platform. 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 BitDiamond Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

BitDiamond

QuillAudits

Canada, India, Singapore and United Kingdom

audits.quillhash.com

hello@quillhash.com