Okratech Token

Smart Contract Security Audit

Prepared by ShellBoxes

Jan 14th, 2022 - Jan 18th, 2022

Shellboxes.com

contact@shellboxes.com

Document Properties

Client	Okratech
Version	1.0
Classification	Public

Scope

The Okratech Token Contract in the Okratech Token Repository

Contract Name	Contract Address
Ortcoin Contract	0x9e711221b34a2d4b8f552bd5f4a6c4e7934920f7

Contacts

COMPANY	EMAIL
ShellBoxes	contact@shellboxes.com

Contents

1 Introduction

Okratech engaged ShellBoxes to conduct a security assessment on the Okratech Token beginning on Jan 14th, 2022 and ending Jan 18th, 2022. In this report, we detail our methodical approach to evaluate potential security issues associated with the implementation of smart contracts, by exposing possible semantic discrepancies between the smart contract code and design document, and by recommending additional ideas to optimize the existing code. Our findings indicate that the current version of smart contracts can still be enhanced further due to the presence of many security and performance concerns.

This document summarizes the findings of our audit.

1.1 About Okratech

Okratech is a DeFi powered and self-governing DAO designed to act as a revolutionary decentralized and broad platform for freelancing. Not only does it provide the highest quality experience for B2B (Business to Business) but also for P2P (Peer to Peer) interactions. Through its intuitive user interface, Ort will match skilled freelancers for job postings across the globe. The highlight of the platform is the absence of transaction fees. Ort's innovative model assures professional mediation ensures both the employer and employee, with the highest quality of work.

Issuer	Okratech
Website	`https://ortcoin.org`
Type	Solidity Smart Contract
Audit Method	Whitebox

1.2 Approach & Methodology

ShellBoxes used a combination of manual and automated security testing to achieve a balance between efficiency, timeliness, practicability, and correctness within the audit's scope. While manual testing is advised for identifying problems in logic, procedure, and implementation, automated testing techniques help to expand the coverage of smart contracts and can quickly detect code that does not comply with security best practices.

1.2.1 Risk Methodology

Vulnerabilities or bugs identified by ShellBoxes are ranked using a risk assessment technique that considers both the LIKELIHOOD and IMPACT of a security incident. This framework is effective at conveying the features and consequences of technological vulnerabilities.

Its quantitative paradigm enables repeatable and precise measurement, while also revealing the underlying susceptibility characteristics that were used to calculate the Risk scores. A risk level will be assigned to each vulnerability on a scale of 5 to 1, with 5 indicating the greatest possibility or impact.

- Likelihood quantifies the probability of a certain vulnerability being discovered and exploited in the untamed.
- Impact quantifies the technical and economic costs of a successful attack.
- Severity indicates the risk's overall criticality.

Probability and impact are classified into three categories: H, M, and L, which correspond to high, medium, and low, respectively. Severity is determined by probability and impact and is categorized into four levels, namely Critical, High, Medium, and Low.

Impact				
	High	Critical	High	Medium
	Medium	High	Medium	Low
	Low	Medium	Low	Low
		High	Medium	Low
			Likelihood	

2 Findings Overview

2.1 Summary

The following is a synopsis of our conclusions from our analysis of the Okratech Token implementation. During the first part of our audit, we examine the smart contract source code and run the codebase via a static code analyzer. The objective here is to find known coding problems statically and then manually check (reject or confirm) issues highlighted by the tool. Additionally, we check business logics, system processes, and DeFi-related components manually to identify potential hazards and/or defects.

2.2 Key Findings

In general, these smart contracts are well-designed and constructed, but their implementation might be improved by addressing the discovered flaws, which include , 2 low-severity vulnerabilities.

Vulnerabilities	Severity	Status
Approve Race	LOW	Acknowledged
Renounce Ownership	LOW	Acknowledged

3 Finding Details

A Okratech.sol

A.1 Approve Race [LOW]

Description:

The standard ERC20 implementation contains a widely-known racing condition in its approve function, wherein a spender is able to witness the token owner broadcast a transaction altering their approval and quickly sign and broadcast a transaction using transferFrom to move the current approved amount from the owner's balance to the spender. If the spender's transaction is validated before the owner's, the spender will be able to get both approval amounts of both transactions.

Code:

Listing 1: Ortcoin.sol

```
function approve(address spender, uint256 amount) external returns (bool
    ↪ ) {
    _approve(_msgSender(), spender, amount);
    return true;
}
```

Risk Level:

Likelihood – 1
Impact - 3

Recommendation:

Use the approve function for the first approval, then use the increaseAllowance and decreaseAllowance functions in order to override the allowance value.

Status - Acknowledged

The Okratech team has acknowledged the risk, stating that the issue is not likely to occur.

A.2 Renounce Ownership [LOW]

Description:

Typically, the contract's owner is the account that deploys the contract. As a result, the owner is able to perform certain privileged activities on his behalf. The renounceOwnership function is used in smart contracts to renounce ownership. Otherwise, if the contract's ownership has not been transferred previously, it will never have an Owner, which can cause a denial of service.

Code:

Listing 2: Ortcoin.sol

```
contract BEP20Token is Context, IBEP20, Ownable {
    using SafeMath for uint256
```

Risk Level:

Likelihood – 1
Impact - 3

Recommendation:

It is advised that the Owner cannot call renounceOwnership without first transferring ownership to a different address. Additionally, if a multi-signature wallet is utilized, executing the renounceOwnership method for two or more users should be confirmed. Alternatively, the renounce ownership functionality can be disabled by overriding it.

Status - Acknowledged

The Okratech team has acknowledged the risk, stating that the issue is not likely to occur, and the team will make sure to avoid this special case when changing the ownership.

4 Static Analysis (Slither)

Description:

ShellBoxes expanded the coverage of the specific contract areas using automated testing methodologies. Slither, a Solidity static analysis framework, was one of the tools used. Slither was run on all-scoped contracts in both text and binary formats. This tool can be used to test mathematical relationships between Solidity instances statically and variables that allow for the detection of errors or inconsistent usage of the contracts' APIs throughout the entire codebase.

Results:

```
BEP20Token.allowance(address,address).owner (Okratech.sol#423) shadows:
        - Ownable.owner() (Okratech.sol#301-303) (function)
BEP20Token._approve(address,address,uint256).owner (Okratech.sol#578)
    ↪ shadows:
        - Ownable.owner() (Okratech.sol#301-303) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #local-variable-shadowing

BEP20Token._burn(address,uint256) (Okratech.sol#557-563) is never used
    ↪ and should be removed
BEP20Token._burnFrom(address,uint256) (Okratech.sol#592-595) is never
    ↪ used and should be removed
Context._msgData() (Okratech.sol#117-120) is never used and should be
    ↪ removed
SafeMath.div(uint256,uint256) (Okratech.sol#216-218) is never used and
    ↪ should be removed
SafeMath.div(uint256,uint256,string) (Okratech.sol#231-238) is never
    ↪ used and should be removed
SafeMath.mod(uint256,uint256) (Okratech.sol#251-253) is never used and
    ↪ should be removed
```

```
SafeMath.mod(uint256,uint256,string) (Okratech.sol#266-269) is never
    ↪ used and should be removed
SafeMath.mul(uint256,uint256) (Okratech.sol#191-203) is never used and
    ↪ should be removed
SafeMath.sub(uint256,uint256) (Okratech.sol#162-164) is never used and
    ↪ should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #dead-code

Redundant expression "this (Okratech.sol#118)" inContext (Okratech.sol
    ↪ #108-121)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #redundant-statements

BEP20Token.constructor() (Okratech.sol#355-363) uses literals with too
    ↪ many digits:
        - _totalSupply = 900000000 * 10 ** 8 (Okratech.sol#359)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #too-many-digits

renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (Okratech.sol#320-323)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (Okratech.sol#329-331)
increaseAllowance(address,uint256) should be declared external:
        - BEP20Token.increaseAllowance(address,uint256) (Okratech.sol
            ↪ #469-472)
decreaseAllowance(address,uint256) should be declared external:
        - BEP20Token.decreaseAllowance(address,uint256) (Okratech.sol
            ↪ #488-491)
mint(uint256) should be declared external:
        - BEP20Token.mint(uint256) (Okratech.sol#501-504)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #public-function-that-could-be-declared-external
```

```
Okratech.sol analyzed (5 contracts with 78 detectors), 18 result(s)
    ↪ found
```

Conclusion:

Most of the vulnerabilities found by the analysis have already been addressed by the smart contract code review.

5 Conclusion

In this audit, we examined the design and implementation of Okratech Token contracts and discovered several issues of low severity. Okratech team has acknowledged all the issues raised in the initial report. Shellboxes' auditors advised Okratech Team to maintain a high level of vigilance and to keep those findings in mind in order to avoid any future complications.

SHELLBOXES

For a Contract Audit, contact us at contact@shellboxes.com