# OKURU-XOT

## Smart Contract Review

**Deliverable: Smart Contract Final Audit Report**

**Security Report**

**October 2022**

# Disclaimer

The information and views set out in this publication are those of the author(s) and do not necessarily reflect the official opinion of the Company. The content, conclusions and recommendations set out in this publication are elaborated in the specific for only project.

eNebula Solutions does not guarantee the authenticity of the project or organization or team of members that is connected/owner behind the project or nor accuracy of the data included in this study. All representations, warranties, undertakings and guarantees relating to the report are excluded, particularly concerning – but not limited to – the qualities of the assessed projects and products. Neither the Company nor any personating on the Company's behalf may be held responsible for the use that may be made of the information contained herein.

eNebula Solutions retains the right to display audit reports and other content elements as examples of their work in their portfolio and as content features in other projects with protecting all security purpose of customer. The report containing confidential information can be used internally by the Customer, or it can be disclosed publicly after all vulnerabilities fixed - upon a decision of the Customer.

# Report Summary

| | | | |
|---|---|---|---|
| **Title** | **OKURU-XOT Smart Contract Audit** | | |
| **Project Owner** | OKURU-XOT | | |
| | | | |
| **Type** | Public | | |
| **Reviewed by** | Vatsal Raychura | **Revision date** | 05/10/2022 |
| **Approved by** | eNebula Solutions Private Limited | **Approval date** | 05/10/2022 |
| | | **№ Pages** | **21** |

# Overview

## Background

OKURU-XOT's team requested that eNebula Solutions perform an Extensive Smart Contract audit of their 'Staking' Smart Contract.

## Project Dates

The following is the project schedule for this review and report:

- **September 25**: Smart Contract Review Completed *(Completed)*
- **September 25**: Delivery of Smart Contract Audit Report *(Completed)*
- **October 05**: Delivery of Final Smart Contract Audit Report *(Completed)*

## Review Team

The following eNebula Solutions team member participated in this review:

- Sejal Barad, Security Researcher and Engineer
- Vatsal Raychura, Security Researcher and Engineer

# Coverage

## Target Specification and Revision

For this audit, we performed research, investigation, and review of the smart contract of OKURU-XOT.

The following documentation repositories were considered in-scope for the review:

- OKURU-XOT Project:

Staking_update.sol

# Introduction

Given the opportunity to review OKURU-XOT Project's smart contract source code, we in the report outline our systematic approach to evaluate potential security issues in the smart contract implementation, expose possible semantic inconsistencies between smart contract code and design document, and provide additional suggestions or recommendations for improvement. Our results show that the given version of smart contracts is ready to launch after resolving the mentioned issues, there are no critical or high issues found related to business logic, security or performance.

About OKURU-XOT: -

| Item | Description |
|---|---|
| **Issuer** | OKURU-XOT |
| **Type** | Staking |
| **Platform** | Solidity |
| **Audit Method** | Whitebox |
| **Latest Audit Report** | October 05, 2022 |

The Test Method Information: -

| Test method | Description |
|---|---|
| **Black box testing** | Conduct security tests from an attacker's perspective externally. |
| **Grey box testing** | Conduct security testing on code modules through the scripting tool, observing the internal running status, mining weaknesses. |
| **White box testing** | Based on the open-source code, non-open-source code, to detect whether there are vulnerabilities in programs such as nodes, SDK, etc. |

The vulnerability severity level information:

| Level | Description |
|---|---|
| **Critical** | Critical severity vulnerabilities will have a significant effect on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities. |
| **High** | High severity vulnerabilities will affect the normal operation of the DeFi project. It is strongly recommended to fix high-risk vulnerabilities. |
| **Medium** | Medium severity vulnerability will affect the operation of the DeFi project. It is recommended to fix medium-risk vulnerabilities. |
| **Low** | Low severity vulnerabilities may affect the operation of the DeFi project in certain scenarios. It is suggested that the project party should evaluate and consider whether these vulnerabilities need to be fixed. |
| **Weakness** | There are safety risks theoretically, but it is extremely difficult to reproduce in engineering. |

The Full List of Check Items:

| Category | Check Item |
|---|---|
| **Basic Coding Bugs** | Constructor Mismatch |
| | Ownership Takeover |
| | Redundant Fallback Function |
| | Overflows & Underflows |
| | Reentrancy |
| | MONEY-Giving Bug |
| | Blackhole |
| | Unauthorized Self-Destruct |
| | Revert DoS |
| | Unchecked External Call |
| | Gasless Send |
| | Send Instead of Transfer |
| | Costly Loop |
| | (Unsafe) Use of Untrusted Libraries |
| | (Unsafe) Use of Predictable Variables |
| | Transaction Ordering Dependence |
| | Deprecated Uses |
| **Semantic Consistency Checks** | Semantic Consistency Checks |
| | Business Logics Review |

| | |
|---|---|
| **Advanced DeFi Scrutiny** | Functionality Checks |
| | Authentication Management |
| | Access Control & Authorization |
| | Oracle Security |
| | Digital Asset Escrow |
| | Kill-Switch Mechanism |
| | Operation Trails & Event Generation |
| | ERC20 Idiosyncrasies Handling |
| | Frontend-Contract Integration |
| | Deployment Consistency |
| | Holistic Risk Management |
| **Additional Recommendations** | Avoiding Use of Variadic Byte Array |
| | Using Fixed Compiler Version |
| | Making Visibility Level Explicit |
| | Making Type Inference Explicit |
| | Adhering To Function Declaration Strictly |
| | Following Other Best Practices |

Common Weakness Enumeration (CWE) Classifications Used in This Audit:

| Category | Summary |
|---|---|
| **Configuration** | Weaknesses in this category are typically introduced during the configuration of the software. |
| **Data Processing Issues** | Weaknesses in this category are typically found in functionality that processes data. |
| **Numeric Errors** | Weaknesses in this category are related to improper calculation or conversion of numbers. |
| **Security Features** | Weaknesses in this category are concerned with topics like authentication, access control, confidentiality, cryptography, and privilege management. (Software security is not security software.) |
| **Time and State** | Weaknesses in this category are related to the improper management of time and state in an environment that supports simultaneous or near-simultaneous computation by multiple systems, processes, or threads. |
| **Error Conditions, Return Values, Status Codes** | Weaknesses in this category include weaknesses that occur if a function does not generate the correct return/status code, or if the application does not handle all possible return/status codes that could be generated by a function. |
| **Resource Management** | Weaknesses in this category are related to improper management of system resources. |

| | |
|---|---|
| **Behavioral Issues** | Weaknesses in this category are related to unexpected behaviors from code that an application uses. |
| **Business Logics** | Weaknesses in this category identify some of the underlying problems that commonly allow attackers to manipulate the business logic of an application. Errors in business logic can be devastating to an entire application. |
| **Initialization and Cleanup** | Weaknesses in this category occur in behaviors that are used for initialization and breakdown. |
| **Arguments and Parameters** | Weaknesses in this category are related to improper use arguments or parameters within function calls. |
| **Expression Issues** | Weaknesses in this category are related to incorrectly written expressions within code. |
| **Coding Practices** | Weaknesses in this category are related to coding practices that are deemed unsafe and increase the chances that an ex pilotable vulnerability will be present in the application. They may not directly introduce a vulnerability, but indicate the product has not been carefully developed or maintained. |

# Findings

## Summary

Here is a summary of our findings after analyzing the OKURU-XOT's Smart Contract. During the first phase of our audit, we studied the smart contract source code and ran our in-house static code analyzer through the Specific tool. The purpose here is to statically identify known coding bugs, and then manually verify (reject or confirm) issues reported by tool. We further manually review business logics, examine system operations, and place DeFi-related aspects under scrutiny to uncover possible pitfalls and/or bugs.

| Severity | No. of Issues |
|---|---|
| Critical | 0 |
| High | 0 |
| Medium | 0 |
| Low | 2(Resolved) |
| Total | 2 |

We have so far identified that there are potential issues with severity of **0 Critical, 0 High, 0 Medium, and 2 Low**. Overall, these smart contracts are well- designed and engineered.

## Functional Overview

| | |
|---|---|
| ($) = payable function<br># = non-constant function | [Pub] public<br>[Ext] external<br>[Prv] private<br>[Int] internal |

+ Context
  - [Int] _msgSender
  - [Int] _msgData

+ Ownable (Context)
  - [Pub] <Constructor> #
  - [Pub] owner
  - [Int] _checkOwner
  - [Pub] renounceOwnership #
    - modifiers: onlyOwner
  - [Pub] transferOwnership #
    - modifiers: onlyOwner
  - [Int] _transferOwnership #

+ Pausable (Context)
  - [Pub] <Constructor> #
  - [Pub] paused
  - [Int] _requireNotPaused
  - [Int] _requirePaused
  - [Int] _pause #
    - modifiers: whenNotPaused
  - [Int] _unpause #

```
    - modifiers: whenPaused

+  ReentrancyGuard
  - [Pub] <Constructor> #
  - [Prv] _nonReentrantBefore #
  - [Prv] _nonReentrantAfter #

+ [Int] IERC20
  - [Ext] totalSupply
  - [Ext] balanceOf
  - [Ext] transfer #
  - [Ext] allowance
  - [Ext] approve #
  - [Ext] transferFrom #

+  Stake (Pausable, Ownable, ReentrancyGuard)
  - [Pub] <Constructor> #
  - [Ext] unstaketoken #
  - [Ext] getTokenExpiry
  - [Ext] stakeToken #
    - modifiers: whenNotPaused
  - [Ext] pause #
    - modifiers: onlyOwner
  - [Ext] unpause #
    - modifiers: onlyOwner
  - [Pub] AddRewardToken #
    - modifiers: onlyOwner
```

# Detailed Results

## Issues Checking Status

### 1. State Variable Default Visibility

- SWC ID: 108
- Severity: Low
- Location: Staking.sol
- Relationship: CWE-710: Improper Adherence to Coding Standards
- Description: State variable visibility is not set. It is best practice to set the visibility of state variables explicitly. The default visibility for "XOTToken", "rewardAmount", "_balances" is internal. Other possible visibility settings are public and private.

```
178  contract Stake is Pausable, Ownable, ReentrancyGuard {
179      IERC20 XOTToken;
180      //  Days (16 * 30 * 24 * 60 * 60)
181      uint256 public Duration = 41472000;
182      uint8 public interestRate = 25 ;
183      uint8 public totalStakers;
184      uint256 rewardAmount ;
185      mapping (address => uint256 ) _balances;
186      struct StakeInfo {
187          uint256 startTS;
188          uint256 endTS;
189          uint256 amount;
190          uint256 claimed;
191      }
```

- Remediations: Variables can be specified as being public, internal or private. Explicitly define visibility for all state variables.
- Resolved: After the first phase of audit, this issue was discussed with the OKURU-XOT's dev team, and they've resolved it before deploying on the chain.

## 2. Code With No Effects

- SWC ID: 135
- Severity: Low
- Location: Staking.sol
- Relationship: CWE-1164: Irrelevant Code
- Description: Usage of equality comparison instead of assignment. This equality comparison doesn't have any effect. Did you mean to do assignment instead?

```
204     function claimReward() external returns (bool){
205         require(addressStaked[_msgSender()] == true, "You are not participated");
206         require(stakeInfos[_msgSender()].endTS < block.timestamp, "Stake Time is not over yet");
207         require(stakeInfos[_msgSender()].claimed == 0, "Already claimed");
208         uint256 stakeAmount = stakeInfos[_msgSender()].amount;
209         uint256 totalTokens = stakeAmount + (stakeAmount * interestRate / 1000);
210         stakeInfos[_msgSender()].claimed == totalTokens;
211         XOTToken.transfer(_msgSender(), totalTokens);
212         emit Claimed(_msgSender(), totalTokens);
213         return true;
214     }
```

- Remediations: It's important to carefully ensure that your contract works as intended. Write unit tests to verify correct behaviour of the code.
- Resolved: After the first phase of audit, this issue was discussed with the OKURU-XOT's dev team, and they've resolved it before deploying on the chain.

## Automated Tools Results

Slither: -

```
Stake.constructor(IERC20,uint256) (Staking.sol#196-203) ignores return value by XOTToken.transferFrom(msg.sender,address(this),amount) (Staking.sol#199)
Stake.claimReward() (Staking.sol#204-214) ignores return value by XOTToken.transfer(_msgSender(),totalTokens) (Staking.sol#211)
Stake.stakeToken(uint256) (Staking.sol#219-234) ignores return value by XOTToken.transferFrom(_msgSender(),address(this),stakeAmount) (Staking.sol#223)
Stake.transferreward(uint256) (Staking.sol#241-245) ignores return value by XOTToken.transferFrom(msg.sender,address(this),rewardAmount) (Staking.sol#243)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer

Reentrancy in Stake.stakeToken(uint256) (Staking.sol#219-234):
        External calls:
        - XOTToken.transferFrom(_msgSender(),address(this),stakeAmount) (Staking.sol#223)
        State variables written after the call(s):
        - addressStaked[_msgSender()] = true (Staking.sol#225)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1

Reentrancy in Stake.constructor(IERC20,uint256) (Staking.sol#196-203):
        External calls:
        - XOTToken.transferFrom(msg.sender,address(this),amount) (Staking.sol#199)
        State variables written after the call(s):
        - _balances[address(this)] = rewardAmount (Staking.sol#201)
        - rewardAmount = amount (Staking.sol#200)
        - totalStakers = 0 (Staking.sol#202)
Reentrancy in Stake.stakeToken(uint256) (Staking.sol#219-234):
        External calls:
        - XOTToken.transferFrom(_msgSender(),address(this),stakeAmount) (Staking.sol#223)
        State variables written after the call(s):
        - stakeInfos[_msgSender()] = StakeInfo(block.timestamp,block.timestamp + Duration,stakeAmount,0) (Staking.sol#226-231)
        - totalStakers ++ (Staking.sol#224)
Reentrancy in Stake.transferreward(uint256) (Staking.sol#241-245):
        External calls:
        - XOTToken.transferFrom(msg.sender,address(this),rewardAmount) (Staking.sol#243)
        State variables written after the call(s):
        - _balances[address(this)] = rewardAmount (Staking.sol#244)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
Reentrancy in Stake.claimReward() (Staking.sol#204-214):
        External calls:
        - XOTToken.transfer(_msgSender(),totalTokens) (Staking.sol#211)
        Event emitted after the call(s):
        - Claimed(_msgSender(),totalTokens) (Staking.sol#212)
Reentrancy in Stake.stakeToken(uint256) (Staking.sol#219-234):
        External calls:
        - XOTToken.transferFrom(_msgSender(),address(this),stakeAmount) (Staking.sol#223)
        Event emitted after the call(s):
        - Staked(_msgSender(),stakeAmount) (Staking.sol#233)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

Stake.claimReward() (Staking.sol#204-214) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(stakeInfos[_msgSender()].endTS < block.timestamp,Stake Time is not over yet) (Staking.sol#206)
        - require(bool,string)(stakeInfos[_msgSender()].claimed == 0,Already claimed) (Staking.sol#207)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

Stake.claimReward() (Staking.sol#204-214) compares to a boolean constant:
        -require(bool,string)(addressStaked[_msgSender()] == true,You are not participated) (Staking.sol#205)
Stake.getTokenExpiry() (Staking.sol#215-218) compares to a boolean constant:
        -require(bool,string)(addressStaked[_msgSender()] == true,You are not participated) (Staking.sol#216)
Stake.stakeToken(uint256) (Staking.sol#219-234) compares to a boolean constant:
        -require(bool,string)(addressStaked[_msgSender()] == false,You already participated) (Staking.sol#221)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality

Context._msgData() (Staking.sol#7-9) is never used and should be removed
ReentrancyGuard._nonReentrantAfter() (Staking.sol#96-100) is never used and should be removed
ReentrancyGuard._nonReentrantBefore() (Staking.sol#88-94) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version0.8.9 (Staking.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Variable Stake.XOTToken (Staking.sol#179) is not in mixedCase
Variable Stake.Duration (Staking.sol#181) is not in mixedCase
Variable Stake._balances (Staking.sol#185) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
Stake.Duration (Staking.sol#181) should be constant
Stake.interestRate (Staking.sol#182) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant

renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (Staking.sol#27-29)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (Staking.sol#30-33)
transferreward(uint256) should be declared external:
        - Stake.transferreward(uint256) (Staking.sol#241-245)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

MythX: -

Report for Staking.sol
https://dashboard.mythx.io/#/console/analyses/43e4093b-83eb-456e-93d2-df8f6abd614a

| Line | SWC Title | Severity | Short Description |
|---|---|---|---|
| 179 | (SWC-108) State Variable Default Visibility | Low | State variable visibility is not set. |
| 184 | (SWC-108) State Variable Default Visibility | Low | State variable visibility is not set. |
| 185 | (SWC-108) State Variable Default Visibility | Low | State variable visibility is not set. |
| 209 | (SWC-101) Integer Overflow and Underflow | Unknown | Arithmetic operation "+" discovered |
| 209 | (SWC-101) Integer Overflow and Underflow | Unknown | Arithmetic operation "*" discovered |
| 209 | (SWC-101) Integer Overflow and Underflow | Unknown | Arithmetic operation "/" discovered |
| 210 | (SWC-135) Code With No Effects | Low | Usage of equality comparison instead of assignment |
| 224 | (SWC-101) Integer Overflow and Underflow | Unknown | Arithmetic operation "++" discovered |
| 228 | (SWC-101) Integer Overflow and Underflow | Unknown | Arithmetic operation "+" discovered |
| 242 | (SWC-101) Integer Overflow and Underflow | Unknown | Arithmetic operation "+" discovered |

Mythril: -

```
root@sv-VirtualBox:/home/sv/Okuru-XOT# myth analyze Staking.sol
The analysis was completed successfully. No issues were detected.
```

Solhint: -

Linter results:

```
Staking.sol:2:1: Error: Compiler version 0.8.16 does not satisfy the r semver requirement

Staking.sol:14:5: Error: Explicitly mark visibility in function (Set ignoreConstructors to true if using
solidity >=0.7.0)

Staking.sol:44:5: Error: Explicitly mark visibility in function (Set ignoreConstructors to true if using
solidity >=0.7.0)

Staking.sol:79:5: Error: Explicitly mark visibility in function (Set ignoreConstructors to true if using
solidity >=0.7.0)

Staking.sol:179:5: Error: Explicitly mark visibility of state

Staking.sol:179:12: Error: Variable name must be in mixedCase

Staking.sol:181:20: Error: Variable name must be in mixedCase

Staking.sol:184:5: Error: Explicitly mark visibility of state

Staking.sol:185:5: Error: Explicitly mark visibility of state

Staking.sol:196:5: Error: Explicitly mark visibility in function (Set ignoreConstructors to true if using
solidity >=0.7.0)

Staking.sol:206:50: Error: Avoid to make time-based decisions in your business logic

Staking.sol:227:26: Error: Avoid to make time-based decisions in your business logic

Staking.sol:228:24: Error: Avoid to make time-based decisions in your business logic
```

## Basic Coding Bugs

1. **Constructor Mismatch**
   - Description: Whether the contract name and its constructor are not identical to each other.
   - Result: PASSED
   - Severity: Critical
2. **Ownership Takeover**
   - Description: Whether the set owner function is not protected.
   - Result: PASSED
   - Severity: Critical
3. **Redundant Fallback Function**
   - Description: Whether the contract has a redundant fallback function.
   - Result: PASSED
   - Severity: Critical
4. **Overflows & Underflows**
   - Description: Whether the contract has general overflow or underflow vulnerabilities
   - Result: PASSED
   - Severity: Critical
5. **Reentrancy**
   - Description: Reentrancy is an issue when code can call back into your contract and change state, such as withdrawing ETHs.
   - Result: PASSED
   - Severity: Critical
6. **MONEY-Giving Bug**
   - Description: Whether the contract returns funds to an arbitrary address.
   - Result: PASSED
   - Severity: High

7. **Blackhole**

   - Description: Whether the contract locks ETH indefinitely: merely in without out.
   - Result: PASSED
   - Severity: High

8. **Unauthorized Self-Destruct**

   - Description: Whether the contract can be killed by any arbitrary address.
   - Result: PASSED
   - Severity: Medium

9. **Revert DoS**

   - Description: Whether the contract is vulnerable to DoS attack because of unexpected revert.
   - Result: PASSED
   - Severity: Medium

10. **Unchecked External Call**

    - Description: Whether the contract has any external call without checking the return value.
    - Result: PASSED
    - Severity: Medium

11. **Gasless Send**

    - Description: Whether the contract is vulnerable to gasless send.
    - Result: PASSED
    - Severity: Medium

12. **Send Instead of Transfer**

    - Description: Whether the contract uses send instead of transfer.
    - Result: PASSED
    - Severity: Medium

13. **Costly Loop**

- Description: Whether the contract has any costly loop which may lead to Out-Of-Gas exception.
- Result: PASSED
- Severity: Medium

14. **(Unsafe) Use of Untrusted Libraries**

- Description: Whether the contract use any suspicious libraries.
- Result: PASSED
- Severity: Medium

15. **(Unsafe) Use of Predictable Variables**

- Description: Whether the contract contains any randomness variable, but its value can be predicated.
- Result: PASSED
- Severity: Medium

16. **Transaction Ordering Dependence**

- Description: Whether the final state of the contract depends on the order of the transactions.
- Result: PASSED
- Severity: Medium

17. **Deprecated Uses**

- Description: Whether the contract use the deprecated tx.origin to perform the authorization.
- Result: PASSED
- Severity: Medium

**Semantic Consistency Checks**

- Description: Whether the semantic of the white paper is different from the implementation of the contract.
- Result: PASSED
- Severity: Critical

## Conclusion

In this audit, we thoroughly analyzed OKURU-XOT's 'Staking' Smart Contract. The current code base is well organized but there are promptly some low-level issues found in the first phase of Smart Contract Audit. After the first phase of audit, this issues were discussed with the OKURU-XOT's dev team, and they've resolved it before deploying on the chain.

Meanwhile, we need to emphasize that smart contracts as a whole are still in an early, but exciting stage of development. To improve this report, we greatly appreciate any constructive feedbacks or suggestions, on our methodology, audit findings, or potential gaps in scope/coverage.

# About eNebula Solutions

We believe that people have a fundamental need to security and that the use of secure solutions enables every person to more freely use the Internet and every other connected technology. We aim to provide security consulting service to help others make their solutions more resistant to unauthorized access to data & inadvertent manipulation of the system. We support teams from the design phase through the production to launch and surely after.

The eNebula Solutions team has skills for reviewing code in C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, and JavaScript for common security vulnerabilities & specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture, including in cryptocurrency, blockchains, payments, and smart contracts. Additionally, the team can utilize various tools to scan code & networks and build custom tools as necessary.

Although we are a small team, we surely believe that we can have a momentous impact on the world by being translucent and open about the work we do.

For more information about our security consulting,
please mail us at – contact@enebula.in