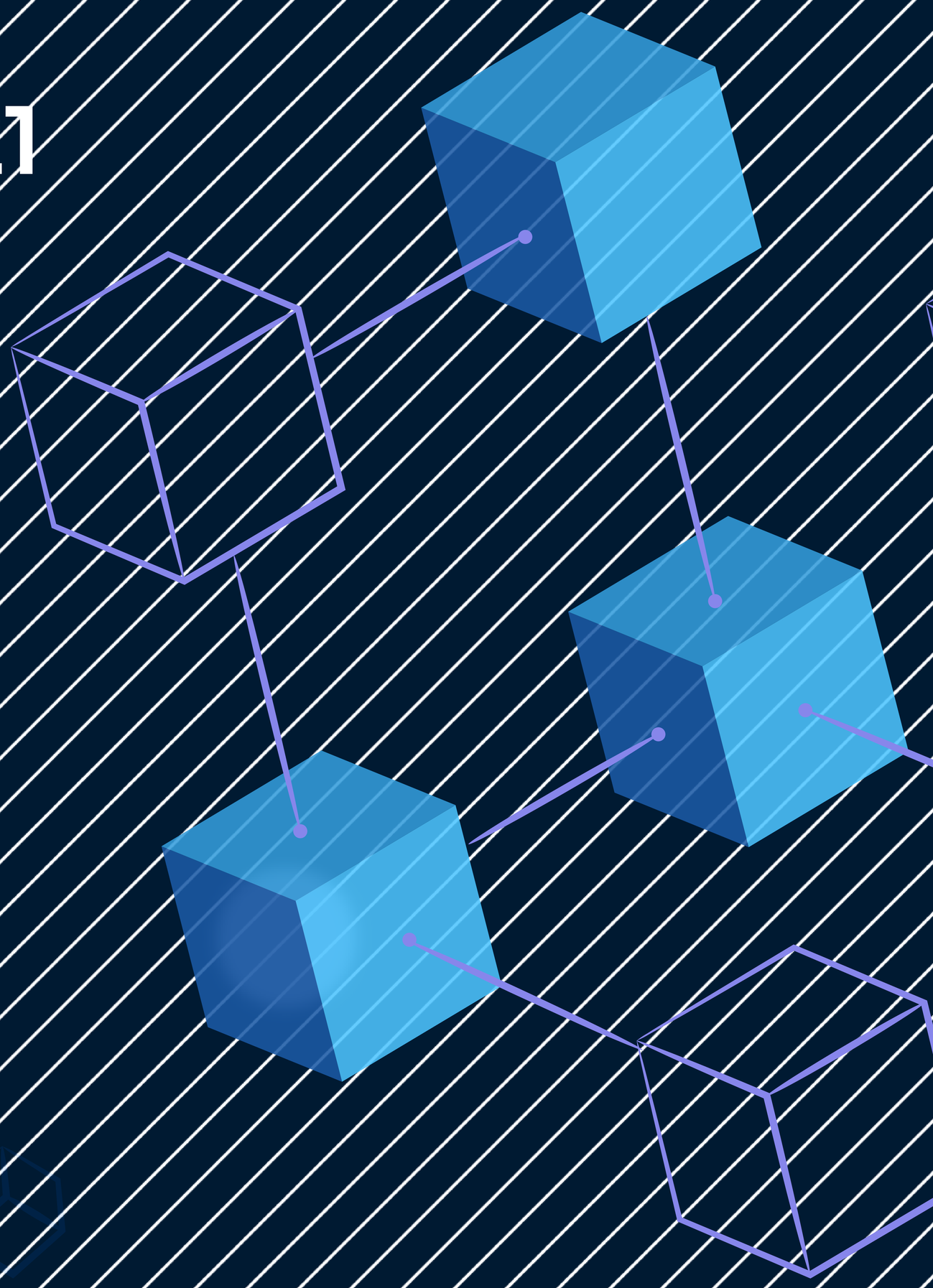
QuillAudits

Audit Report
November, 2021



For

Xololnu

Contents

Contents

Scope of the Audit

The scope of this audit was to analyze and document the XoloInu Token smart contract codebase for quality, security, and correctness.

Checked Vulnerabilities

We have scanned the smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered:

- 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 contract, 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 analysed 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
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 analysed, 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 behaviour 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
Remix IDE, Truffle, Truffle Team, Solhint, Mythril, Slither, Solidity statistic analysis.

Issue Categories

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.

Risk-level	Description
High	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	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	Low-level severity issues can cause minor impact and or 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 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.

Number of issues per severity

Type	High	Medium	Low	Informational
Open	0	0	0	0
Acknowledged	1	0	1	20
Closed	0	0	1	0

Introduction

During the period of **November 13, 2021 to November 15, 2021** - QuillAudits Team performed a security audit for **XoloInu** smart contracts.

The code for the audit was taken from following the official link:
Codebase: https://github.com/dopa-admin/Xolo-Inu

Note	Date	Commit hash	Branch
Version 1	November	1732ddc19ff59ad9f4ed38c13236862be0b7b037	Master
Version 2	November	f8f7afcb460585bbfa60a67e96cabfed4a80b33e	Master

Issues Found

A. Contract – XoloInu

High severity issues

1. Contract code size exceeds 24576 bytes

Description
Contract implementation is too large in size to be deployed on mainnet. Ethereum with its spurious dragon release limited the size of the contracts deployable on mainnet to 24576 bytes.
The size of the contract XoloInu.sol goes way above this value and currently is of size 28055 bytes

Remediation
Define and use libraries for pure and view functions e.g. We can create a library which contains all the mathematical operations.

Status: Acknowledged
As the deployment is planned on BSC so the contract can be deployed successfully.

Medium severity issues

No issues were found.

Low level severity issues

2. Use of block.timestamp for comparisons

Description
The value of block.timestamp can be manipulated by the miner.
And conditions with strict equality is difficult to achieve -
block.timestamp == launchTime

Remediation
Avoid use of block.timestamp

Status: Acknowledged

3. Missing Zero Address Validation

Description
Function **_transfer():** Missing Zero Address Check for **to** address

Remediation
Add a ‘require’ to check **to** address != address(0)

Status: Closed
The implementation is done in the new commit.

Informational issues

4. Missing comments and description

Description
Comments and Description of the methods and the variables are missing, it's hard to read and understand the purpose of the variables and the methods in context of the whole picture

Remediation
Consider adding NatSpec format comments for the comments and state variables

Status: Acknowledged

5. Incorrect and Inconsistent use of Indentations and Spaces

Description
Throughout the contract there are several lines which have not been indented properly in consistency with the other lines of code.
There are several extra spaces present in the empty lines and also added after and in between the characters in a particular line

Remediation
Trim all the extra spaces and indent all the lines correctly

Status: Acknowledged

6. **Less meaningful variable and method names**

Description
Certain variables and method names does not provide clear picture of their purpose

Remediation
- **MAX** should be renamed to **MAX_INT_256**
- **'inSwapAndLiquify'** can be simply named **'_locked'**
- **"newBalance"** in **swapAndLiquify** should be called **"addedBalance"** or
- **"swappedEth"**
- **"takeMarketing"** should be called **"takeMarketingFee"**

Status: Acknowledged

7. **Incorrect method names**

Description
Incorrect names used on the method caused confusion while usage and can be problematic in the future.

Function **isRemovedSniper** returns whether the passed address isSniper or not.

Function **_removeSniper** is adding setting isSniper to for the passed address

Remediation
- **isRemovedSniper** should be renamed to **isSniper**
- **_removeSniper** should be renamed to **addSnipe**

Status: Acknowledged

8. **Variable defined but never used**

Description
'deadAddress' defined but never used

Remediation
Remove unused variables

Status: Acknowledged

9. **Public methods only being used externally**

Description
‘public’ functions that are never by the contract should be declared ‘external’ to save gas.

Remediation
Make these methods external - **setSwapAndLiquifyEnabled, setRouterAddress, isRemovedSniper, includeInFee, excludeFromFee, isExcludedFromFee, excludeFromReward, reflectionFromToken, deliver, totalFees, isExcludedFromReward**

Status: Acknowledged

10. Constant calculations in the contract

Description
Precalculated initialization will save 2847 units of gas in deployment

```
uint256 private _tTotal = 10000000000 * 10**18;
uint256 private _rTotal = (MAX - (MAX % _tTotal));
```

Recommendation
Replace the initialization as -

uint256 private _tTotal = 10000000000000000000000000000000; // 10000000000 * 1018; // 10000000000 * 10**18;**

uint256 private _rTotal = 115792089237316195423570985008687907853269984665640000000000000000000000000000 // (MAX - (MAX % _tTotal));

10**2 should be a precalculated constant and then used in **calculateTaxFee** and **calculateLiquidityFee** to save more than 10 units on gas on every call

Status: Acknowledged

11. **Use constructor to set addresses**

    Description
    Address being set in contract should be provided dynamically via constructor on deployment for readability and correctness

    Remediation
    Pass the address of **“marketingWallet”** and **“uniswapV2Router”** as a parameter of the constructor and set the value

    Status: Acknowledged

12. **Reading address(this) multiple times**

    Description
    Methods **constructor, _takeLiquidity, swapAndLiquify, swapTokensForEth, addLiquidity** and **takeMarketing** are reading and typecasting several times in the same block costing extra gas.

    Remediation
    Define a local address variable and use it instead to avoid repeated type casting

    Status: Acknowledged

13. **Getters should be at the bottom of the contract**

    Status: Acknowledged

14. **Length calculation within the loop**

    Description
    Reading from the state and fetching its length is a costly operation and doing such within a loop should be avoided

    Remediation
    _excluded.length should be calculated and stored in a variable before using in the for loop inside `**_getCurrentSupply**` method

    Status: Acknowledged

15. Ordering conditions correctly can reduce boolean operations

```
function reflectionFromToken(uint256 tAmount, bool deductTransferFee) public view returns(uint256) {
    require(tAmount <= _tTotal, "Amount must be less than supply");
    if (!deductTransferFee) {
        (uint256 rAmount,,,,,) = _getValues(tAmount);
        return rAmount;
    } else {
        (,uint256 rTransferAmount,,,,) = _getValues(tAmount);
        return rTransferAmount;
    }
}
```

Remediation
By recording the if condition we can reduce one boolean operation and increase readability

```
if (deductTransferFee) {
        (,uint256 rTransferAmount,,,,) = _getValues(tAmount);
        return rTransferAmount;
    } else {
        (uint256 rAmount,,,,,) = _getValues(tAmount);
        return rAmount;
    }
```

Status: Acknowledged

16. Inconsistent error messages

Description
The require checks in the contract have error messages containing the contract name in some places and the same has been skipped in most cases.

Remediation
The error message should follow the same pattern and we recommend using the contract name along with the error message at all places
E.g. **require(!_isExcluded[account], "XoloInu: Account is already excluded");**

Status: Acknowledged

17. Keeping the rarest condition check first saves gas in cases when it is followed by other condition checks

```
// buy
      if(from == uniswapV2Pair && to != address(uniswapV2Router) && !
_isExcludedFromFee[to]) {
          require(tradingOpen, "Trading not yet enabled.");

          //antibot
          if (block.timestamp == launchTime) {
            _isSniper[to] = true;
            _confirmedSnipers.push(to);
          }
      }
```

Remediation

```
require(tradingOpen, "Trading not yet enabled.");
      if(block.timestamp == launchTime && from == uniswapV2Pair && to !=
address(uniswapV2Router) && !_isExcludedFromFee[to]) {
          _isSniper[to] = true;
          _confirmedSnipers.push(to);
      }
```

Status: Acknowledged

18. Redundant calculations

```
// split the Liquidity tokens balance into halves
      uint256 half = tokensForLiquidity.div(2);
      uint256 otherHalf = tokensForLiquidity.sub(half);
```

Description

Here both the variables will always have the same value but we are calculating the same value twice and using it in two different variables as well.

Remediation

```
uint256 half = tokensForLiquidity.div(2);
      uint256 otherHalf = half;
```

We can also avoid creating another variable and use the variable **half** twice

Status: Acknowledged

19. Naming Conventions

Description
The contract follows a consistent naming convention where we are private variables with leading “_” and public variables without it.
But we have missed to comply to the condition for certain variable names - **"transferToAddressETH"** and **"takeMarketing"** which are private and **“_removeSniper”** which is external

Remediation
Remove “_” from external variable names and add it to private variable names

Status: Acknowledged

20. Same execution for two different cases can be merged

```
if (_isExcluded[sender] && !_isExcluded[recipient]) {
            _transferFromExcluded(sender, recipient, amount);
        } else if (!_isExcluded[sender] && _isExcluded[recipient]) {
            _transferToExcluded(sender, recipient, amount);
        } else if (!_isExcluded[sender] && !_isExcluded[recipient]) {
            _transferStandard(sender, recipient, amount);
        } else if (_isExcluded[sender] && _isExcluded[recipient]) {
            _transferBothExcluded(sender, recipient, amount);
        } else {
            _transferStandard(sender, recipient, amount);
        }
```

Remediation

```
if (_isExcluded[sender] && !_isExcluded[recipient]) {
            _transferFromExcluded(sender, recipient, amount);
        } else if (!_isExcluded[sender] && _isExcluded[recipient]) {
            _transferToExcluded(sender, recipient, amount);
        } else if (_isExcluded[sender] && _isExcluded[recipient]) {
            _transferBothExcluded(sender, recipient, amount);
        } else {
            _transferStandard(sender, recipient, amount);
        }
```

_transferStandard has been moved to else block and one if case is reduced

Status: Acknowledged

21. **Avoid hardcoded addresses**

```
require(account != 0x10ED43C718714eb63d5aA57B78B54704E256024E, 'We can not blacklist Uniswap');
```

Remediation

use already defined uniswapV2Router for comparison

```
require(account != address(uniswapV2Router), 'We can not blacklist Uniswap');
```

Status: Acknowledged

22. **Unnecessary use of SafeMath**

Solidity version 0.8 was released with safeMath checks inbuilt, we can avoid using an explicit safe math library

Status: Acknowledged

23. **Misleading comment**

“_tokenTransfer” has a misleading comment about a variable called “takeFee” while there is no such variable defined in the contract

Status: Acknowledged

Closing Summary

Overall, smart contracts are very well written, documented and adhere to guidelines. Several issues of High and Low severity have been reported and all have been explained and the relevant ones are covered in the new commit hash. **The contract can be deployed to a public BSC chain**.

Disclaimer

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

Audit Report
November, 2021

For

Xololnu

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com