verichains

SECURITY AUDIT OF

QUBE TOKEN

Public Report

Jul 23, 2024

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
Ethereum	An open source platform based on blockchain technology to create and distribute smart contracts and decentralized applications.
Ether (ETH)	A cryptocurrency whose blockchain is generated by the Ethereum platform. Ether is used for payment of transactions and computing services in the Ethereum network.
Smart contract	A computer protocol intended to digitally facilitate, verify or enforce the negotiation or performance of a contract.
Solidity	A contract-oriented, high-level language for implementing smart contracts for the Ethereum platform.
Solc	A compiler for Solidity.
ERC20	ERC20 (BEP20 in Binance Smart Chain or *x*RP20 in other chains) tokens are blockchain-based assets that have value and can be sent and received. The primary difference with the primary coin is that instead of running on their own blockchain, ERC20 tokens are issued on a network that supports smart contracts such as Ethereum or Binance Smart Chain.

EXECUTIVE SUMMARY

This Security Audit Report was prepared by Verichains Lab on Jul 23, 2024. We would like to thank the Qube for trusting Verichains Lab in auditing smart contracts. Delivering high-quality audits is always our top priority.

This audit focused on identifying security flaws in code and the design of the Qube Token. The scope of the audit is limited to the source code files provided to Verichains. Verichains Lab completed the assessment using manual, static, and dynamic analysis techniques.

During the audit process, the audit team had identified some vulnerable issues in the source code, along with some recommendations. Qube team has acknowledged and resolved some of these issues.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY 5

1.1. About Qube Token 5

1.2. Audit Scope 5

1.3. Audit Methodology 6

1.4. Disclaimer 7

1.5. Acceptance Minute 7

2. AUDIT RESULT 8

2.1. Overview 8

2.2. Findings 8

2.2.1. Fee-on-transfer token should not be used with Uniswap V3 MEDIUM 9

2.2.2. Incorrect token balance handling in `_burn` function LOW 10

2.2.3. Possible of integer overflow in `increaseAllowance` function LOW 11

2.2.4. Token is vulnerable to front-running approval attacks INFORMATIVE 11

2.2.5. Unnecessary complex logic when configuring parameters INFORMATIVE 12

3. VERSION HISTORY 14

1. MANAGEMENT SUMMARY

1.1. About Qube Token

In the deep reaches of space lies a galaxy buzzing with excitement — welcome to Sky Rangers! Here, brave explorers known as Rangers journey across the cosmos in search of a remarkable energy source called **Qube**. These shimmering cubes hold the power to transform everything they touch.

In Sky Rangers, your quest begins with Qube, your ticket to thrilling adventures across the cosmos. Use Qube to embark on epic quests where each step brings you closer to incredible discoveries. The more Qube you collect, the further you can explore, uncovering hidden treasures and facing daring challenges along the way.

Earn as you play and watch your fun and fortune grow in the enchanting journey of Sky Rangers!

- ZKL: Swap ZKL tokens for gold credits, your key to unlocking new adventures.
- Gold Credits: Use these credits to power up your gameplay and access exciting games.
- Games: Join thrilling games to have fun and earn a fortune.
- Blind Boxes: Earn in-game blind boxes and open them to reveal rare NFTs.
- NFTs: Deploy digital characters as NFTs to supercharge your computing power.
- Hash Power: Convert game activity into hash power to fuel your mining efforts.
- **Qube**: Mine Qube tokens, the ultimate treasure that strengthens your game presence and gains value as you hold them.

Qube: Your Galactic Companion

- Limited Edition: There's a finite amount of Qube, making each one a precious gem.
- Why It Matters: Powered by blockchain technology, every Qube transaction is secure and transparent, ensuring trust in every galactic deal.
- A Share for All: A portion of each Qube transaction goes into a community pool, allowing every Sky Ranger to earn daily rewards.
- Staying Stellar: We carefully manage Qube circulation to maintain its value, ensuring it remains a cherished asset in the vast expanse of Sky Rangers.

1.2. Audit Scope

This audit focused on identifying security flaws in code and the design of the Qube Token. It was conducted on commit `88443762bb40c1dcb2c630e02fc79643a282de01` from git repository `https://github.com/fcwrsmall/solidity.qube`.

1.3. Audit Methodology

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and RK87, our in-house smart contract security analysis tool.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that were considered during the audit of the smart contract:

- Integer Overflow and Underflow
- Timestamp Dependence
- Race Conditions
- Transaction-Ordering Dependence
- DoS with (Unexpected) revert
- DoS with Block Gas Limit
- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Reentrancy
- Explicit visibility of functions state variables (external, internal, private and public)
- Logic Flaws

For vulnerabilities, we categorize the findings into categories as listed in table below, depending on their severity level:

SEVERITY LEVEL	DESCRIPTION
CRITICAL	A vulnerability that can disrupt the contract functioning; creates a critical risk to the contract; required to be fixed immediately.
HIGH	A vulnerability that could affect the desired outcome of executing the contract with high impact; needs to be fixed with high priority.
MEDIUM	A vulnerability that could affect the desired outcome of executing the contract with medium impact in a specific scenario; needs to be fixed.
LOW	An issue that does not have a significant impact, can be considered as less important.

Table 1. Severity levels

1.4. Disclaimer

Qube acknowledges that the security services provided by Verichains, are conducted to the best of their professional abilities but cannot guarantee 100% coverage of all security vulnerabilities. Qube understands and accepts that despite rigorous auditing, certain vulnerabilities may remain undetected. Therefore, Qube agrees that Verichains shall not be held responsible or liable, and shall not be charged for any hacking incidents that occur due to security vulnerabilities not identified during the audit process.

1.5. Acceptance Minute

This final report served by Verichains to the Qube will be considered an Acceptance Minute. Within 7 days, if no any further responses or reports is received from the Qube, the final report will be considered fully accepted by the Qube without the signature.

2. AUDIT RESULT

2.1. Overview

The Qube Token was written in `Solidity` language, with the required version to be in range `>=0.8.0 <0.9.0`.

The contract is a custom ERC20 token with fixed-capacity, burnable, and fee-on-transfer features. The `balanceOf` function is customized to work with the Uniswap V3 liquidity pool. Most of the contract's configuration properties can be set by the owner, such as the LP address, buy/sell fee tax ratios, buy/sell enable flags, etc.

The total supply of the token is fixed at 100 million QUBE tokens, which are minted to the owner's address upon deployment. The table below describes some properties of the audited Qube Token (as of the report writing time).

PROPERTY	VALUE
Name	QUBE
Symbol	QUBE
Decimals	18
Total Supply	$100{,}000{,}000 \times 10^{18}$ (it represents 100 million tokens)

Table 2. The Qube Token properties

2.2. Findings

During the audit process, the audit team found some vulnerabilities and recommendations in the given version of Qube Token.

#	Issue	Severity	Status
1	Fee-on-transfer token should not be used with Uniswap V3	MEDIUM	Acknowledged
2	Incorrect token balance handling in `_burn` function	LOW	Fixed
3	Possible of integer overflow in `increaseAllowance` function	LOW	Fixed

#	Issue	Severity	Status
4	Token is vulnerable to front-running approval attacks	INFORMATIVE	Fixed
5	Unnecessary complex logic when configuring parameters	INFORMATIVE	Acknowledged

2.2.1. Fee-on-transfer token should not be used with Uniswap V3 MEDIUM

Affected files:

- contracts/fi-zklink/qube.sol

The Qube token is a fee-on-transfer token that charges a tax fee on every transfer, which is not compatible with Uniswap V3. Modifying the `balanceOf` function to bypass the tax fee check is not recommended, as this can cause the token to become incompatible with the current DeFi ecosystem (the `block.number` should not affect the balance).

For example, after tokens are sent from the LP to the `recipient`, any other protocol that depends on the `balanceOf(recipient)` in the same block will get an incorrect result.

```
function balanceOf(address account) override external view virtual returns(uint256) {
    unchecked {
        Account memory a = _accounts[account];      /// a workaround for some V3 checking
        return (uint48(block.number) == a.blockNo) ? a.blockView : a.balance; // AUDIT: NOT
RECOMMENDED
    }
}

function _beforeTransfer(uint256 balance, address from, address to, uint256 amount)
internal virtual returns(uint256) {
    unchecked {
        require(balance >= amount,'$');
        if(from != msg.sender) {                    /// non-owner caller must check
allowance
            uint256 allowed = _allowances[from][msg.sender];
            if(allowed < MAX256) {
                require(allowed >= amount,'%');
                _allowances[from][msg.sender] = allowed-amount;
            }
        }
        if(to == address(this))                     /// tokens sent to this contract will
be treated as donation to 'fi'
            return amount;
        address fi = _fi;
        if((to == fi)||(from == fi))                /// no restriction for 'fi' contract
            return amount;
```

```
        Defi memory defi = _defi;
        uint256 tax = (from == defi.lp) ? _getCommission(defi.enables,defi.ppmBuy,amount) :
                        (to == defi.lp) ?
_getCommission(defi.enables>>1,defi.ppmSell,amount) :
                                          0;
        if(tax == 0)
            return amount;
        _afterTransfer(from,address(this),tax,tax);
        return amount-tax;
    }
}
```

RECOMMENDATION

The tax fee should be collected from the LP pool instead of the transfer function. This will make the token compatible with all other DeFi protocols, and less likely to cause issues in the future.

By the way, customizing the transfer logic by splitting it into multiple functions like `_beforeTransfer` and `_afterTransfer` in this case is quite complex and error-prone. Specifically, calling `_afterTransfer` for tax inside `_beforeTransfer` does not make any sense and dramatically reduces the readability of the code. It is recommended to simplify or refactor the transfer logic to avoid potential issues.

UPDATES

- **Jul 23, 2024**: This issue has been acknowledged by the Qube team.

2.2.2. Incorrect token balance handling in `_burn` function LOW

Affected files:

- contracts/fi-zklink/qube.sol

In the `_burn` function, the balance of the zero address is increased by the amount of tokens to be burned despite the decrease in the total supply. This will result in a mismatch between the total supply and the sum of all addresses' balances.

```
function _burn(address from, uint256 amount) internal virtual returns(bool) {
    unchecked {
        uint256 total = _total;
        if(total < amount)
            return false;
        _total = total-amount;
        _accounts[_0].balance += uint104(amount); // AUDIT: SHOULD BE REMOVED
        emit Transfer(from,_0,amount);
        return true;
    }
}
```

RECOMMENDATION

Burning by transferring the tokens to the zero address should only be used when the burn function is not supported by the token. In this case, we should remove the line that increases the balance of the zero address.

UPDATES

- **Jul 23, 2024**: This issue has been acknowledged and fixed by the Qube team.

2.2.3. Possible of integer overflow in `increaseAllowance` function LOW

Affected files:

- contracts/fi-zklink/qube.sol

With the current declared Solidity version, the `increaseAllowance` function may be vulnerable to integer overflow. The `addedValue` parameter is added to the current allowance value, which can cause an overflow if the sum exceeds the maximum value of a `uint256`.

```
pragma solidity >=0.7.0 <0.9.0;

function increaseAllowance(address to, uint256 addedValue) override external virtual
returns(bool) {
    return _approve(msg.sender,to,_allowances[msg.sender][to]+addedValue);
}
```

RECOMMENDATION

The Solidity version declaration statement should be updated to `pragma solidity >=0.8.0 <0.9.0` to prevent integer overflow.

UPDATES

- **Jul 23, 2024**: This issue has been acknowledged and fixed by the Qube team.

2.2.4. Token is vulnerable to front-running approval attacks INFORMATIVE

Affected files:

- contracts/fi-zklink/qube.sol

The `approve` functions for Qube token do not have any mechanism to protect the logic against front-running attacks. For more information about front-running attack related to the `approve` function, refer to the following document.

https://docs.google.com/document/d/1YLPtQxZu1UAvO9cZ1O2RPXBbT0mooh4DYKjA_jp-RLM

```
function approve(address to, uint256 amount) override external virtual returns(bool) {
    return _approve(msg.sender,to,amount);
}

function _approve(address from, address to, uint256 amount) internal virtual returns(bool)
{
    _allowances[from][to] = amount;
    emit Approval(from,to,amount);
    return true;
}
```

RECOMMENDATION

Besides having the `increaseAllowance` and `decreaseAllowance` functions, we should also modify the logic of the `approve` function so that it only allows changing the allowance value from/to zero.

UPDATES

- **Jul 23, 2024**: This issue has been acknowledged and fixed by the Qube team.

2.2.5. Unnecessary complex logic when configuring parameters INFORMATIVE

Affected files:

- contracts/fi-zklink/qube.sol

The implementation for configuring parameters in the Qube token contract is unnecessarily complex. The `_config` function is used to set different parameters based on the `op` parameter, which can be simplified by using separate functions for each configuration parameter.

The ownership check logic should be separated using a modifier, which will improve the readability of the code.

Overuse of bitwise operators in `_enable` function just make the function harder to read and understand. It is recommended to use a more straightforward approach to set the enable flags.

```
function _enable(uint8 b, bool enable) internal virtual {
    unchecked {
        if(enable) _defi.enables |= b;
        else _defi.enables &= ~b;
    }
}

function _config(uint256 op, address to, uint256 n) internal virtual returns(bool) {
    unchecked {
        require(msg.sender == _owner,"Denied!");
        if(op == transferOwner_ ) _owner = to; else /// transfer owner, to=0 to renouce
ownership
```

```
        if(op == fi_            ) _fi = to; else
        if(op == lp_            ) _defi.lp = to; else
        if(op == buy_           ) _enable(0x1,n != 0); else
        if(op == sell_          ) _enable(0x2,n != 0); else
        if(op == ppmbuy_        ) _defi.ppmBuy = uint16(n); else
        if(op == ppmsell_       ) _defi.ppmSell = uint16(n); else
            return false;
        return true;
    }
}

function transferOwner(address to) override external virtual {
    if(_config(transferOwner_,to,0))
        emit TransferOwner(msg.sender,to);
}
////////////////////////////////////////////////////////
function configV3lp(address lp) override external virtual {
    _config(lp_,lp,0);
}
////////////////////////////////////////////////////////
function configV3Buy(uint256 ppm) override external virtual {
    _config(ppmbuy_,_0,ppm);
}
////////////////////////////////////////////////////////
function configV3Sell(uint256 ppm) override external virtual {
    _config(ppmsell_,_0,ppm);
}
////////////////////////////////////////////////////////
function enableBuy(bool enable) override external virtual {
    _config(buy_,_0,enable ? 1 : 0);
}
////////////////////////////////////////////////////////
function enableSell(bool enable) override external virtual {
    _config(sell_,_0,enable ? 1 : 0);
}
```

UPDATES

- **Jul 23, 2024**: This issue has been acknowledged by the Qube team.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Jul 23, 2024*	Public Report	Verichains Lab

Table 3. Report versions history