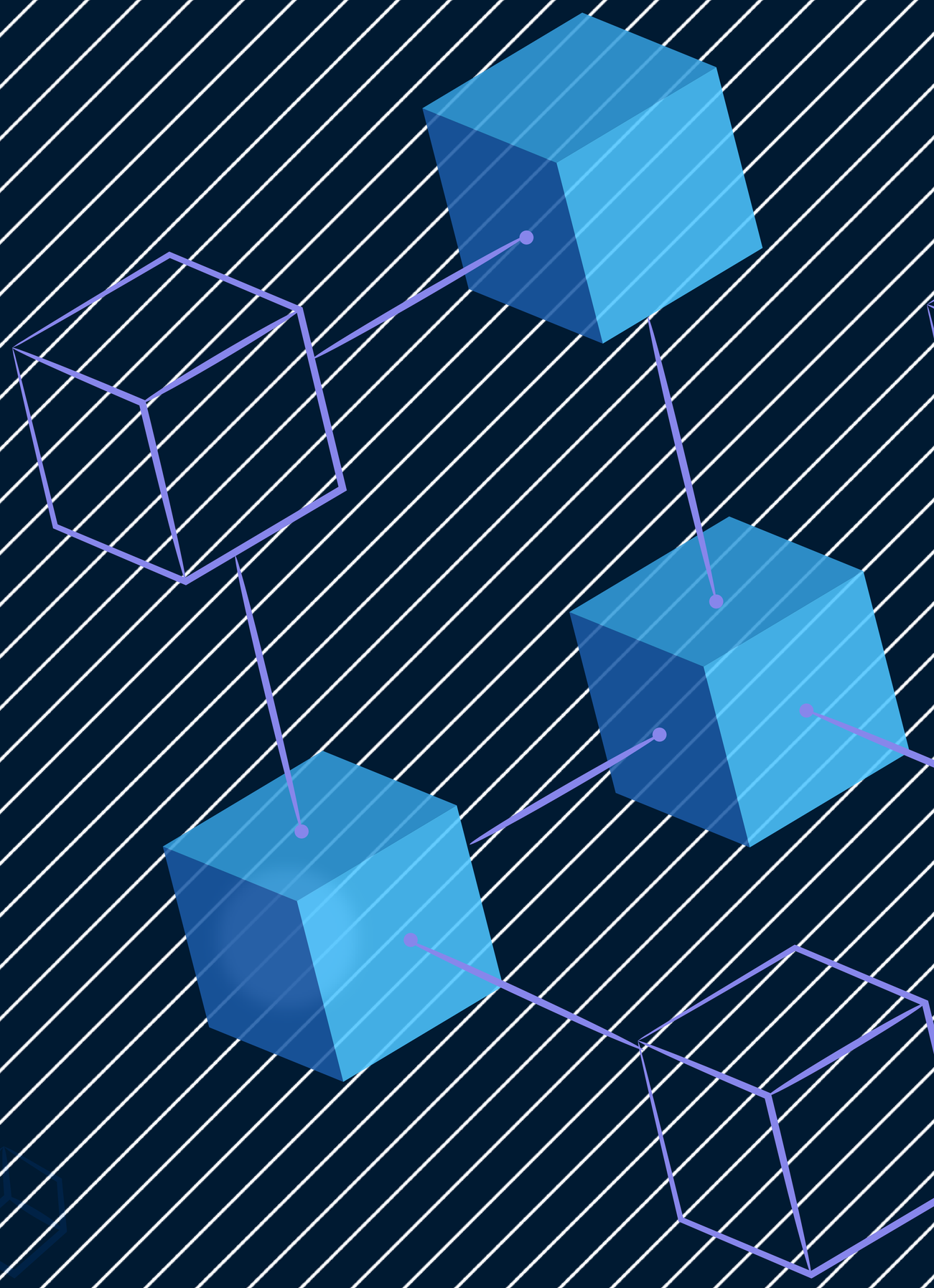
QuillAudits

Audit Report
October, 2021



For

TANGO

Contents

Scope of the Audit

The scope of this audit was to analyze and document the TangoChain 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/BEP-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, Solhint, 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	3
Acknowledged	0	0	0	3
Closed	0	0	0	0

Introduction

From **October 5th, 2021, to October 7, 2021**, the QuillAudits Team performed a security audit for the **TangoChain** smart contract.

The code for the audit was taken from following the link https://bscscan.com/address/0x259c52A12fD6c4457C3d2113c2252B23de55D9A6#code

Issues Found

A. Contract – Tango

High severity issues

No issues were found.

Medium severity issues

No issues were found.

Low severity issues

No issues were found.

Informational issues

1. **Acknowledged**

```
363         uint256 _maxAmountAtStart = 930000000000000000000000000000; // Supply
364         uint256 _disableMaxTxDate;
```

As _maxAmountAtStart is fixed, it can be declared as a constant

2. **Acknowledged**

```
454         if (!_whiteList[recipient] && !_whiteList[sender] && _disableMaxTxDate > 0 && block.timestamp < _disableMaxTxDate) {
455             require(amount <= _maxAmountAtStart, "Big transactions are locked the first 24h");
456         }
```

This check is not required as the total supply of the tokens is always <=_maxAmountAtStart, so for any transfer transaction, the amount will be <=_maxAmountAtStart and whether the user is whitelisted or not, as the amount is always <= _maxAmountAtStart, this require will always succeed
In case there the user puts in more amount then the _maxAmountAtStart, this require fails, but in that case the proper event emitted should be that the **"ERC20: transfer amount exceeds balance"**, As the user has put in an amount more than the total supply of the token itself.

3. Acknowledged

```
438     function addToWhitelist(address wallet) onlyOwner() external {
439         _whiteList[wallet] = true;
440     }
441
442     function removeFromWhitelist(address wallet) onlyOwner() external {
443         _whiteList[wallet] = false;
444     }
```

Lack of events when adding or removing an address from the whitelist.

4. Open

use consistent solidity pragma throughout all the smart contracts

5. Open

```
366     uint256 _disableMaxTxDate;
367
```

```
450     function setMaxTxDate() onlyOwner() external {
451         require(_disableMaxTxDate == 0, "Already setted");
452         _disableMaxTxDate = block.timestamp + 24 hours;
453     }
```

Redundant variable, as it might not be used in the business logic anymore, and the function setMaxTxDate is restricted to the owner and does not affect the functionality of the token so the variable and this function might be redundant.

6. Open

```
440     function addToWhitelist(address wallet) onlyOwner() external {
441         _whiteList[wallet] = true;
442         emit AddedToWhitelist(wallet);
443     }
444
445     function removeFromWhitelist(address wallet) onlyOwner() external {
446         _whiteList[wallet] = false;
447         emit RemovedToWhitelist(wallet);
448     }
449
```

```
363
364     mapping(address => bool) _whiteList;
```

As there is no further use of the whitelisted address in the contract other than this, This could mean that these functions and maps are redundant, unless otherwise some functionality related to this is further added

Functional test

Function Names	Testing results
constructor	PASS
symbol	PASS
name	PASS
totalSupply	PASS
balanceOf	PASS
transfer	PASS
allowance	PASS
approve	PASS
transferFrom	PASS
increaseAllowance	PASS
decreaseAllowance	PASS
addToWhitelist	PASS
removeFromWhitelist	PASS
burn	PASS
burnFrom	PASS

Automated Tests

SOLIDITY STATIC ANALYSIS

Security

Block timestamp:Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree.

That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.morePos: 451:28:

Block timestamp:Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree.

That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.morePos: 457:86:

Gas & Economy

Gas costs:Gas requirement of function Tango.transferOwnership is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage

(this includes clearing or copying arrays in storage)Pos: 333:4:

Gas costs:Gas requirement of function Tango.name is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage

(this includes clearing or copying arrays in storage)Pos: 380:4:

Gas costs:Gas requirement of function Tango.symbol is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage

(this includes clearing or copying arrays in storage)Pos: 384:4:

Gas costs:Gas requirement of function Tango.transfer is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage

(this includes clearing or copying arrays in storage)Pos: 400:4:

Gas costs:Gas requirement of function Tango.approve is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage

(this includes clearing or copying arrays in storage)Pos: 409:4:

Gas costs:Gas requirement of function Tango.transferFrom is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)Pos: 414:4:

Gas costs:Gas requirement of function Tango.increaseAllowance is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)Pos: 420:4:

Gas costs:Gas requirement of function Tango.decreaseAllowance is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)Pos: 425:4:

Gas costs:Gas requirement of function Tango.burn is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)Pos: 430:4:

Gas costs:Gas requirement of function Tango.burnFrom is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)Pos: 434:4:

Miscellaneous

Constant/View/Pure functions:SafeMath.sub(uint256,uint256) : Is constant but potentially should not be.
Note: Modifiers are currently not considered by this static analysis.morePos: 155:4:

Constant/View/Pure functions:SafeMath.div(uint256,uint256) : Is constant but potentially should not be. Note:
Modifiers are currently not considered by this static analysis.morePos: 212:4:

Constant/View/Pure functions:SafeMath.mod(uint256,uint256) : Is constant but potentially should not be.
Note: Modifiers are currently not considered by this static analysis.morePos: 248:4:

Similar variable names:Tango.burnFrom(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.Pos: 435:47:

Similar variable names:Tango.burnFrom(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.Pos: 435:74:

Similar variable names:Tango.burnFrom(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.Pos: 437:17:

Similar variable names:Tango.burnFrom(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.Pos: 438:14:

Similar variable names:Tango.burnFrom(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.Pos: 438:23:

Similar variable names:Tango._burn(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.Pos: 466:16:

Similar variable names:Tango._burn(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.Pos: 467:18:

Similar variable names:Tango._burn(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.Pos: 467:39:

Similar variable names:Tango._burn(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.Pos: 467:52:

Similar variable names:Tango._burn(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.Pos: 468:40:

Similar variable names:Tango._burn(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.Pos: 469:22:

Similar variable names:Tango._burn(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.Pos: 469:43:

Guard conditions:Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.morePos: 140:8:

Guard conditions:Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.morePos: 170:8:

Guard conditions:Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.morePos: 195:8:

Guard conditions:Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.morePos: 229:8:

Guard conditions:Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.morePos: 265:8:

Guard conditions:Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.morePos: 313:8:

Guard conditions:Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.morePos: 334:8:

Guard conditions:Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.morePos: 450:8:

Guard conditions:Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.morePos: 455:8:

Guard conditions:Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.morePos: 456:8:

Guard conditions:Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.morePos: 458:12:

Guard conditions:Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.morePos: 466:8:

Guard conditions:Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.morePos: 473:8:

Guard conditions:Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.morePos: 474:8:

Data truncated:Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.Pos: 195:16:

Data truncated:Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.Pos: 230:20:

Results

No major issues were found. Some false positive errors were reported by the tool. All the other issues have been categorized above according to their level of severity.

Closing Summary

Overall, smart contracts are very well written. No instances of Integer Overflow and Underflow vulnerabilities or Back-Door Entry were found in the contract.

Disclaimer

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

Audit Report
October, 2021

For

TANGO

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com