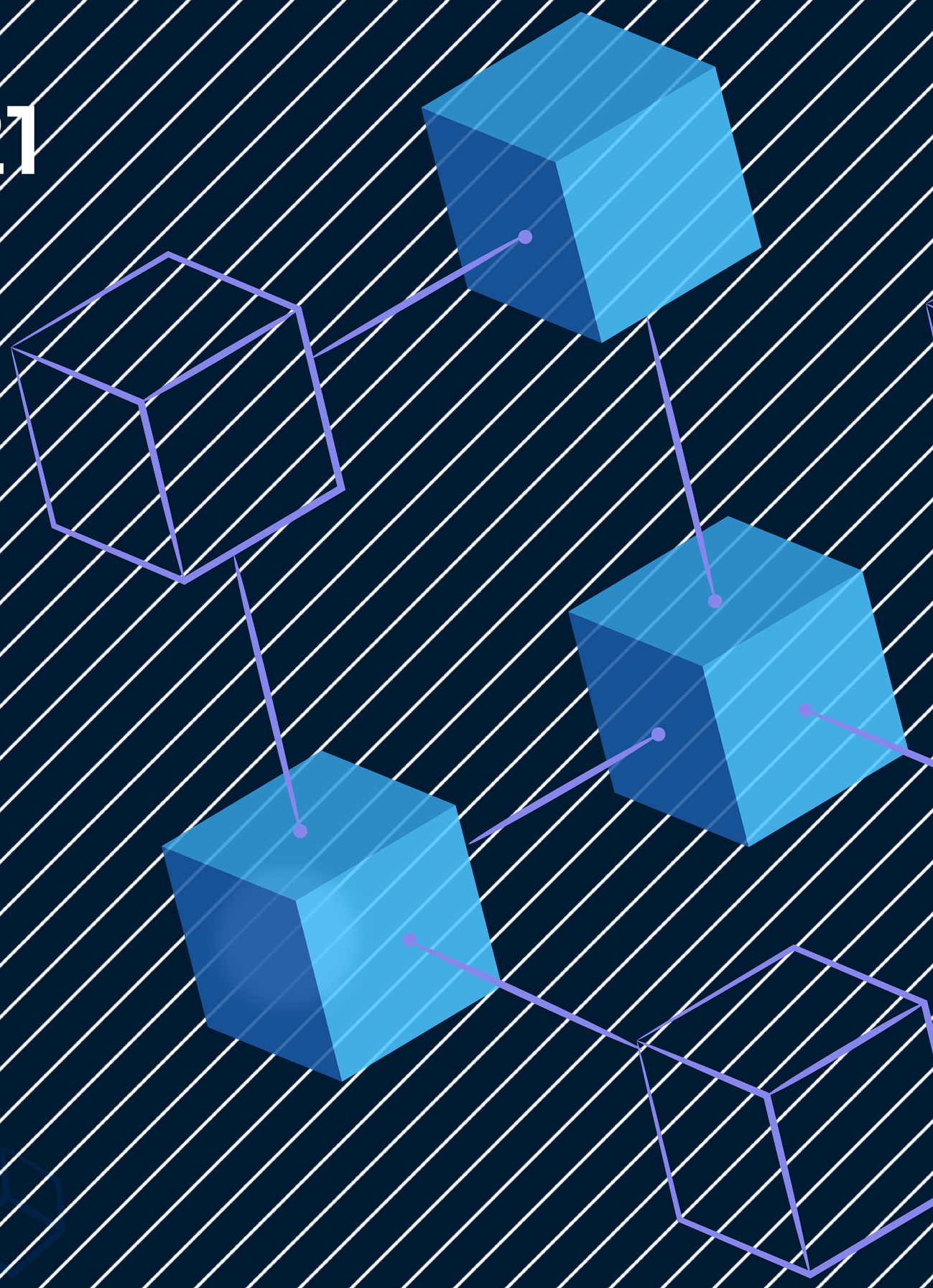
QuillAudits

Audit Report
December, 2021



For

GOGOcoin

Contents

Contents

Scope of the Audit

The scope of this audit was to analyze and document the GoGocoin 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, Theo.

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	0	0	5	3
Closed	0	0	0	0

Introduction

During the period of **October 19, 2021, to November 2, 2021** - QuillAudits Team performed a security audit for **GoGocoin** smart contract.

The code for the audit was taken from the following official File: https://github.com/gogocoin/gogo-contracts

V	Date	Github	Commit ID
1	October	https://github.com/gogocoin/gogo-contracts	f952d59650263cd8e0d403d5a8f3d94890869ba1

Issues Found

A. Contract – GoGo Contracts

High severity issues

No issues were found.

Medium severity issues

No issues were found.

Low severity issues

1. Approve Race

```
2
3   pragma solidity ^0.8.4;
4
```

```
2
3   pragma solidity ^0.8.4;
4
```

Description

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Here in GoGoToken.sol and TokenSale.sol, the current pragma Solidity directive is "^0.8.4". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Remediation

Lock the pragma version and also consider known bugs (https://github.com/ethereum/solidity/releases) for the compiler version that is chosen.

Reference
https://swcregistry.io/docs/SWC-103
https://consensys.github.io/smart-contract-best-practices/recommendations/#lock-pragmas-to-specific-compiler-version

Status: Acknowledged

2. State Variable Default Visibility

```
10  contract TokenSale is Ownable, ReentrancyGuard {
11          uint256 public price;
12          uint256 public maxBuyAmount;
13          uint256 public cap;
14          IERC20 TokenContract;
15          uint256 public tokensSold;
16          address usdcAddress = 0x2791Bca1f2de4661ED88A30C99A7a9449Aa84174;
17          address[] path;
18          uint256 public releaseTime;
19          uint256 public unlockTime;
20          bool public refundable = false;
21          uint256 multiplier = 30;
22
23          IUniswapV2Router02 router =
24                  IUniswapV2Router02(0xa5E0829CaCEd8fFDD4De3c43696c57F7D7A678ff);
```

Description
Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.

Here in the TokenSale.sol It is best practice to set the visibility of state variables explicitly. The default visibility for "TokenContract", "usdcAddress", "path", "multiplier", "router" are internal. Other possible visibility settings are public and private.

Remediation
Variables can be specified as being public, internal or private. Explicitly define visibility for all state variables.

Reference
https://swcregistry.io/docs/SWC-108
https://consensys.github.io/smart-contract-best-practices/recommendations/#explicitly-mark-visibility-in-functions-and-state-variables

Status: Acknowledged

3. Must add a require statement in the constructor to check that the _releaseTime should be greater than block.timestamp.
Because there are high chances of setting incorrect _releaseTime.
require(_releaseTime > block.timestamp, "Release time is before current time");

Status: Acknowledged

4. Must add a require statement in the constructor to check that the _unlockTime should be greater than block.timestamp.
Because there are high chances of setting incorrect _unlockTime.
require(_unlockTime > block.timestamp, "Release time is before current time");

Status: Acknowledged

5. Also add a require statement to make sure whether _releaseTime is lesser than the _unlockTime or not
according to business logic.

Status: Acknowledged

Informational issues

6. It is advised to emit appropriate events after updating the price via setPrice() function, after setting the maxBuyAmount

   Status: Acknowledged

7. declare "usdcAddress" and "multiplier" state variables as constants, as This can help reduce gas costs

   Status: Acknowledged

8. It is advised to set proper values for maticPrice and usdcPrice. There can be an issue of buying the tokens via buy() and buybyUSDC() at a rate less than the actual price set as there is division done in these functions. It should be taken into account that it is possible for gasprices to drop significantly in future.

   Status: Acknowledged

Functional test

Function Names	Testing results
priceinWeis	Passed
setPrice	Passed
setMaxBuyAmount	Passed
buy	Passed
buyByUSDC	Passed
claim	Passed
unLock	Passed
getRefund	Passed
setRefundable	Passed
endSale	Passed

Automated Tests

Slither

No major issues were found. There are just some Low and Informational errors that were reported by the tool.

```
Different versions of Solidity is used:
        - Version used: ['^0.8.0', '^0.8.4']
        - ^0.8.0 (Context.sol#3)
        - ^0.8.0 (ERC20.sol#3)
        - ^0.8.4 (GoGoToken.sol#3)
        - ^0.8.0 (IERC20.sol#3)
        - ^0.8.0 (IERC20Metadata.sol#3)
        - ^0.8.0 (Ownable.sol#3)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

Context._msgData() (Context.sol#20-22) is never used and should be removed
ERC20._burn(address,uint256) (ERC20.sol#274-289) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.0 (Context.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (ERC20.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.4 (GoGoToken.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (IERC20.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (IERC20Metadata.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (Ownable.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.4 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

GoGoToken.constructor() (GoGoToken.sol#9-11) uses literals with too many digits:
        - _mint(owner(),100000000 * (10 ** decimals())) (GoGoToken.sol#10)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
```

```
name() should be declared external:
        - ERC20.name() (ERC20.sol#61-63)
symbol() should be declared external:
        - ERC20.symbol() (ERC20.sol#69-71)
totalSupply() should be declared external:
        - ERC20.totalSupply() (ERC20.sol#93-95)
balanceOf(address) should be declared external:
        - ERC20.balanceOf(address) (ERC20.sol#100-102)
transfer(address,uint256) should be declared external:
        - ERC20.transfer(address,uint256) (ERC20.sol#112-115)
allowance(address,address) should be declared external:
        - ERC20.allowance(address,address) (ERC20.sol#120-122)
approve(address,uint256) should be declared external:
        - ERC20.approve(address,uint256) (ERC20.sol#131-134)
transferFrom(address,address,uint256) should be declared external:
        - ERC20.transferFrom(address,address,uint256) (ERC20.sol#149-163)
increaseAllowance(address,uint256) should be declared external:
        - ERC20.increaseAllowance(address,uint256) (ERC20.sol#177-180)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20.decreaseAllowance(address,uint256) (ERC20.sol#196-204)
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (Ownable.sol#53-55)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (Ownable.sol#61-64)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

```
Reentrancy in TokenSale.getRefund() (TokenSale.sol#169-185):
        External calls:
        - require(bool)(IERC20(usdcAddress).transfer(msg.sender,allocation.usdcInvested)) (TokenSale.sol#176-178)
        External calls sending eth:
        - address(msg.sender).transfer((allocation.nativeAmount + allocationLocked.nativeAmount) * price) (TokenSale.sol#180-182)
        State variables written after the call(s):
        - delete purchasedAmount[msg.sender] (TokenSale.sol#183)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities

Reentrancy in TokenSale.buyByUSDC(uint256) (TokenSale.sol#109-144):
        External calls:
        - require(bool,string)(IERC20(usdcAddress).transferFrom(msg.sender,address(this),amounts[1]),TF: Check allowance) (TokenSale.sol#117-124)
        State variables written after the call(s):
        - allocation.usdcAmount += (_buyAmount * multiplier) / 100 (TokenSale.sol#126)
        - allocation.usdcInvested += amounts[1] (TokenSale.sol#131)
        - tokensSold += _buyAmount (TokenSale.sol#141)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1

TokenSale.setPrice(uint256) (TokenSale.sol#65-67) should emit an event for:
        - price = _newprice (TokenSale.sol#66)
TokenSale.setMaxBuyAmount(uint256) (TokenSale.sol#69-71) should emit an event for:
        - maxBuyAmount = _maxBuyAmount (TokenSale.sol#70)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic

Reentrancy in TokenSale.buyByUSDC(uint256) (TokenSale.sol#109-144):
        External calls:
        - require(bool,string)(IERC20(usdcAddress).transferFrom(msg.sender,address(this),amounts[1]),TF: Check allowance) (TokenSale.sol#117-124)
        State variables written after the call(s):
        - allocationLocked.usdcAmount += (_buyAmount * (100 - multiplier)) / 100 (TokenSale.sol#130)
Reentrancy in TokenSale.getRefund() (TokenSale.sol#169-185):
        External calls:
        - require(bool)(IERC20(usdcAddress).transfer(msg.sender,allocation.usdcInvested)) (TokenSale.sol#176-178)
        External calls sending eth:
        - address(msg.sender).transfer((allocation.nativeAmount + allocationLocked.nativeAmount) * price) (TokenSale.sol#180-182)
        State variables written after the call(s):
        - delete lockedAmount[msg.sender] (TokenSale.sol#184)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2

Reentrancy in TokenSale.buyByUSDC(uint256) (TokenSale.sol#109-144):
        External calls:
        - require(bool,string)(IERC20(usdcAddress).transferFrom(msg.sender,address(this),amounts[1]),TF: Check allowance) (TokenSale.sol#117-124)
        Event emitted after the call(s):
        - Sold(msg.sender,_buyAmount,false) (TokenSale.sol#143)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
```

```
TokenSale.claim() (TokenSale.sol#146-155) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(releaseTime < block.timestamp,Cannot claim before the sale ends) (TokenSale.sol#147-150)
TokenSale.unLock() (TokenSale.sol#157-167) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(unlockTime < block.timestamp,Cannot unlock before the unlock time) (TokenSale.sol#158-161)
TokenSale.getRefund() (TokenSale.sol#169-185) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(releaseTime < block.timestamp,Cannot get refunded before the sale ends) (TokenSale.sol#170-173)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

Different versions of Solidity is used:
        - Version used: ['^0.8.0', '^0.8.4']
        - ^0.8.0 (Context.sol#3)
        - ^0.8.0 (IERC20.sol#3)
        - ^0.8.4 (IUniswapV2Router01.sol#3)
        - ^0.8.4 (IUniswapV2Router02.sol#3)
        - ^0.8.0 (Ownable.sol#3)
        - ^0.8.0 (ReentrancyGuard.sol#3)
        - ^0.8.4 (TokenSale.sol#3)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

Context._msgData() (Context.sol#20-22) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.0 (Context.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (IERC20.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.4 (IUniswapV2Router01.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.4 (IUniswapV2Router02.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (Ownable.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (ReentrancyGuard.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.4 (TokenSale.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.4 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Function IUniswapV2Router01.WETH() (IUniswapV2Router01.sol#8) is not in mixedCase
Parameter TokenSale.setPrice(uint256)._newprice (TokenSale.sol#65) is not in mixedCase
Parameter TokenSale.setMaxBuyAmount(uint256)._maxBuyAmount (TokenSale.sol#69) is not in mixedCase
Parameter TokenSale.buy(uint256)._buyAmount (TokenSale.sol#81) is not in mixedCase
Parameter TokenSale.buyByUSDC(uint256)._buyAmount (TokenSale.sol#109) is not in mixedCase
Parameter TokenSale.setRefundable(bool)._flag (TokenSale.sol#187) is not in mixedCase
Variable TokenSale.TokenContract (TokenSale.sol#14) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Reentrancy in TokenSale.getRefund() (TokenSale.sol#169-185):
        External calls:
        - address(msg.sender).transfer((allocation.nativeAmount + allocationLocked.nativeAmount) * price) (TokenSale.sol#180-182)
        State variables written after the call(s):
        - delete lockedAmount[msg.sender] (TokenSale.sol#184)
        - delete purchasedAmount[msg.sender] (TokenSale.sol#183)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-4

Variable IUniswapV2Router01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amountADesired (IUniswapV2Router01.sol#13) i
s too similar to IUniswapV2Router01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amountBDesired (IUniswapV2Router01.s
ol#14)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-are-too-similar

TokenSale.multiplier (TokenSale.sol#21) should be constant
TokenSale.usdcAddress (TokenSale.sol#16) 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() (Ownable.sol#53-55)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (Ownable.sol#61-64)
priceinWeis() should be declared external:
        - TokenSale.priceinWeis() (TokenSale.sol#61-63)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

Mythx

There are just some Low errors that were reported by the tool.

Report for GoGoToken.sol
https://dashboard.mythx.io/#/console/analyses/137933a1-d8b5-4173-aa07-3919b2f8d185

Line	SWC Title	Severity	Short Description
3	(SWC-103) Floating Pragma	Low	A floating pragma is set.

Report for TokenSale.sol
https://dashboard.mythx.io/#/console/analyses/34b8d69f-2518-4a71-9640-75a7c0f54dd8

Line	SWC Title	Severity	Short Description
3	(SWC-103) Floating Pragma	Low	A floating pragma is set.
14	(SWC-108) State Variable Default Visibility	Low	State variable visibility is not set.
16	(SWC-108) State Variable Default Visibility	Low	State variable visibility is not set.
17	(SWC-108) State Variable Default Visibility	Low	State variable visibility is not set.
21	(SWC-108) State Variable Default Visibility	Low	State variable visibility is not set.
23	(SWC-108) State Variable Default Visibility	Low	State variable visibility is not set.

Mythril

There are just some Low errors that were reported by the tool.

```
root@sv-VirtualBox:/home/sv/Quillhash-Audits# myth analyze GoGoToken.sol
The analysis was completed successfully. No issues were detected.
```

```
root@sv-VirtualBox:/home/sv/Quillhash-Audits# myth analyze TokenSale.sol
 The analysis was completed successfully. No issues were detected.
```

Solhint

There are no errors reported by the tool.

Linter results:

```
GoGoToken.sol:3:1: Error: Compiler version ^0.8.4 does not satisfy the r semver requirement
```

```
GoGoToken.sol:9:2: Error: Explicitly mark visibility in function (Set ignoreConstructors to true if using solidity >=0.7.0)
```

Linter results:

```
TokenSale.sol:3:1: Error: Compiler version ^0.8.4 does not satisfy the r semver requirement
```

```
TokenSale.sol:14:2: Error: Explicitly mark visibility of state
```

```
TokenSale.sol:14:9: Error: Variable name must be in mixedCase
```

```
TokenSale.sol:16:2: Error: Explicitly mark visibility of state
```

```
TokenSale.sol:17:2: Error: Explicitly mark visibility of state
```

```
TokenSale.sol:21:2: Error: Explicitly mark visibility of state
```

```
TokenSale.sol:23:2: Error: Explicitly mark visibility of state
```

```
TokenSale.sol:42:2: Error: Explicitly mark visibility in function (Set ignoreConstructors to true if using
solidity >=0.7.0)
```

```
TokenSale.sol:148:18: Error: Avoid to make time-based decisions in your business logic
```

```
TokenSale.sol:159:17: Error: Avoid to make time-based decisions in your business logic
```

```
TokenSale.sol:171:18: Error: Avoid to make time-based decisions in your business logic
```

Closing Summary

Overall, During the audit, low severity issues associated with the token found. It is recommended to fix these before deployment. Overall the optimization issue is neglectable. Instances of Integer Overflow and Underflow, Reentrancy, or any other major vulnerabilities are not found in the contract.

Disclaimer

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

Audit Report
December, 2021

For

GOGO coin

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com