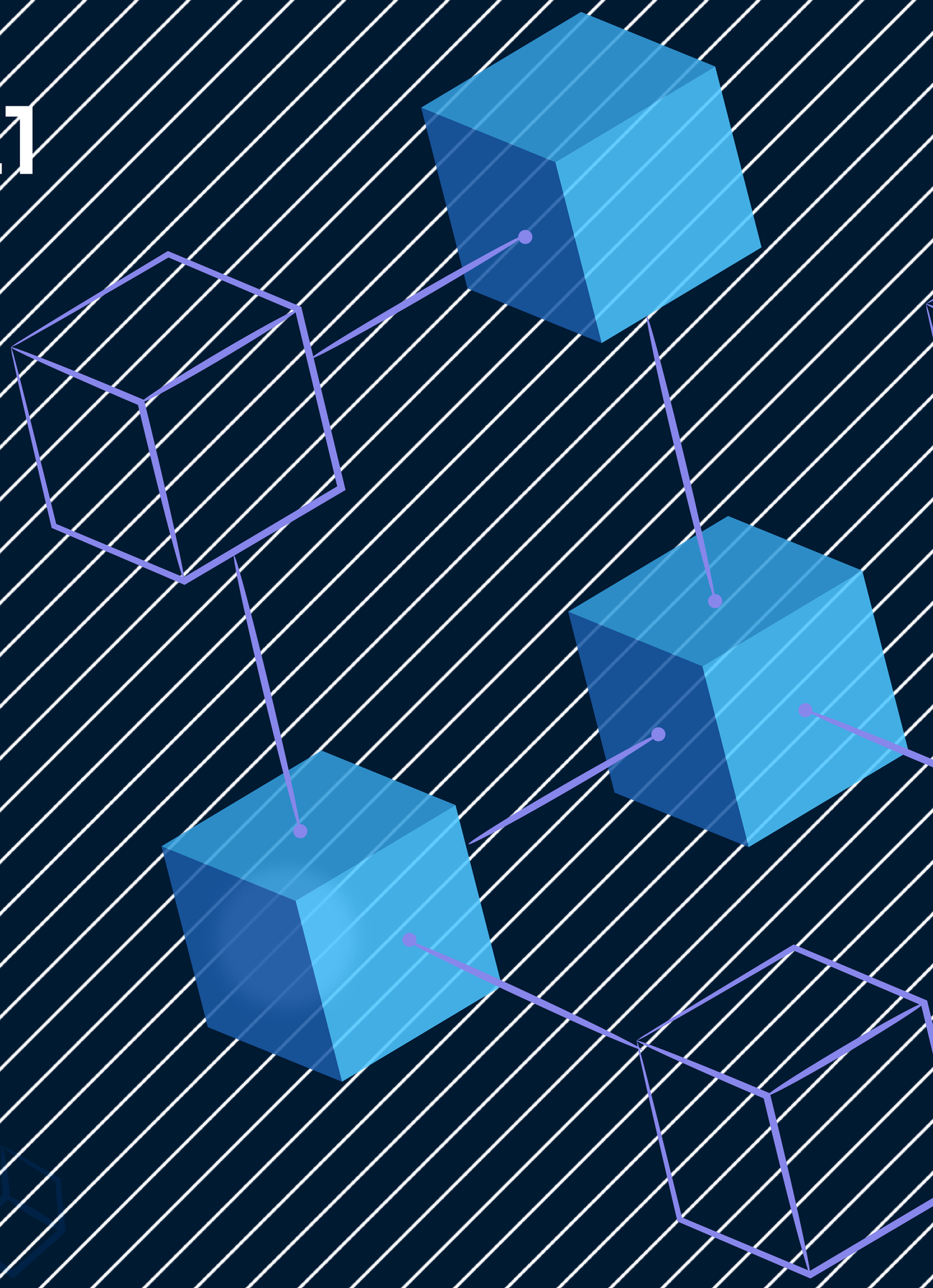
QuillAudits

Audit Report
November, 2021



For

Contents

Contents

Scope of the Audit

The scope of this audit was to analyze and document the Yoda smart contracts 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	3	4	0
Closed	0	0	2	0

Introduction

During the period of **October 16, 2021, to October 26, 2021** - QuillAudits Team performed a security audit for **Yoda Prediction** smart contracts.

The code for the audit was taken from the following official repo of Yoda: https://github.com/yoda-xyz/yoda-prediction-contracts

Note	Date	Commit hash
Version 1	October	47076c50953d932587127f9b684f350fa69fe90d

Issues Found

A. Contract – Factory

High severity issues

No issues were found.

Medium severity issues

A.1 Looping Over Dynamic Array

```
Line 74:
for (uint256 i = 0; i < len; i++) {
            address questionClone = ClonesUpgradeable.clone(questionContract);
            bytes memory payload = abi.encodeWithSelector(0x660b88ee,
                questionData[i].data, adminAddress, operatorAddress);
          (bool success, bytes memory returnData) = address(questionClone).call(payload);
        require(success && (returnData.length == 0 || abi.decode(returnData, (bool))),
            "Initialization Failed");
            questionAddressArray[questionData[i].gameID].push(questionClone);
            questionAddressMap[questionData[i].questionID] = questionClone;
            emit QuestionCreated(
                questionData[i].gameID,
                questionData[i].questionID,
                questionClone,
                questionData.length
            );
        }
```

```
Line 119:
for (uint256 i = 0; i < len; i++) {
            address gameQuestion = questionAddressArray[gameID][i];
            require(gameQuestion != address(0), "Question Does not exist");
            (bool success, bytes memory returnData) = address(gameQuestion).call(payload);
            require(success && (returnData.length == 0 || abi.decode(returnData, (bool))),
           "ClaimGame Failed");
        }
```

```
Line 134:
for (uint256 i = 0; i < len; i++) {
            address gameQuestion = questionAddressArray[gameID][i];
            require(gameQuestion != address(0), "Question Does not exist");
            (bool success, bytes memory returnData) = address(gameQuestion).call(payload);
            require(success && (returnData.length == 0 || abi.decode(returnData, (bool))),
           "ClaimGameTreasury Failed");
        }
```

Description
When smart contracts are deployed or their associated functions are invoked, the execution of these operations always consumes a certain quantity of gas, according to the amount of computation required to accomplish them. Modifying an unknown-size array that grows in size over time can result in a Denial of Service attack. Simply by having an excessively huge array, users can exceed the gas limit, therefore preventing the transaction from ever succeeding.

Remediation
Avoid actions that involve looping across the entire data structure. If you really must loop over an array of unknown size, arrange for it to consume many blocs and thus multiple transactions.

Acknowledged
The Team has acknowledged the risk knowing that this particular function is only be used by the operator and will max have 15 lengths to this looping array.

Low severity issues

A.2 Floating pragma

```
Line 2:
pragma solidity ^0.8.4;
```

Description
The contract makes use of the floating-point pragma 0.8.4 Contracts should be deployed using the same compiler version and flags that were used during the testing process. Locking the pragma helps ensure that contracts are not unintentionally deployed using another pragma, such as an obsolete version that may introduce issues in the contract system.

Remediation
Consider locking the pragma version. It is advised that floating pragma not be used in production. Both truffle-config.js and hardhat.config.js support locking the pragma version.

Acknowledged
The Team has acknowledged the risk

A.3 Renounce Ownership

```
Line 11:
contract Factory is Initializable, UUPSUpgradeable, OwnableUpgradable,
NativeMetaTransaction {
```

Description

Typically, the contract's owner is the account that deploys the contract. As a result, the owner is able to perform certain privileged activities on his behalf. The renounceOwnership function is used in smart contracts to renounce ownership. Otherwise, if the contract's ownership has not been transferred previously, it will never have an Owner, which is risky.

Remediation

It is advised that the Owner cannot call renounceOwnership without first transferring ownership to a different address. Additionally, if a multi-signature wallet is utilized, executing the renounceOwnership method for two or more users should be confirmed. Alternatively, the Renounce Ownership functionality can be disabled by overriding it.

Acknowledged

The Team has acknowledged the risk

A.4 Missing Address Verification

```
Line 26:
function initialize(address _adminAddress, address _operatorAddress) external initializer {
        __Ownable_init();
        _initializeEIP712("Prediction-Market");
        adminAddress = _adminAddress;
        operatorAddress = _operatorAddress;
    }
```

```
Line 63:
function addQuestionType(bytes32 _id, address _questionAddress) external onlyOperator
{
        require(questionType[_id] == address(0), "Implementation Exists");
        questionType[_id] = _questionAddress;
    }
```

Description
Certain functions lack a safety check in the address, the address-type argument should include a zero-address test, otherwise, the contract's functionality may become inaccessible or the token may be burned in perpetuity.

Remediation
It’s recommended to undertake further validation prior to user-supplied data. The concerns can be resolved by utilizing a whitelist technique or a modifier.

Solved
The Team has solved the issue by checking the value of the address with the zero address.

B. Contract – GameQuestion

High severity issues

No issues were found.

Medium severity issues

B.1 Race Condition

```
Line 138:
function setRewardRate(uint8 _rewardRate) external onlyAdmin {
        require(_rewardRate <= TOTAL_RATE, "rewardRate cannot be more than 100%");
        rewardRate = _rewardRate;
        treasuryRate = TOTAL_RATE - _rewardRate;
        emit RatesUpdated(questionID, rewardRate, treasuryRate);
        }
```

```
Line 150:
function setTreasuryRate(uint8 _treasuryRate) external onlyAdmin {
        require(_treasuryRate <= TOTAL_RATE, "treasuryRate cannot be more than 100%");
        rewardRate = TOTAL_RATE - _treasuryRate;
        treasuryRate = _treasuryRate;
        emit RatesUpdated(questionID, rewardRate, treasuryRate);
        }
```

Description
The rewardRate and treasuryRate variables have setters; if the admin modifies one of these values there is a possibility of race condition. A user might use these variables with some specific values and then the admins modify them. It's possible that the admin's transaction gets mined before the user's transaction and that will make the user use the new values of these variables without knowing it.

Remediation
Add additional arguments to the functions that use these variables with their current value, then add a require statement that verifies that the value provided in the arguments are equal to the ones stored in the smart contract.

Acknowledged
The Team has acknowledged the risk knowing that these functions will only be used by the admin after the betting is closed and before execution of the question. So only the latest value will be picked by the contract to calculate the rewards.

B.2 Usage of block.timestamp

```
Line 66:
function _bettable() internal view returns (bool) {
        return
            question.startTime != 0 &&
            question.lockTime != 0 &&
            block.timestamp > question.startTime &&
            block.timestamp < question.lockTime;
        }
```

Description
Block.timestamp is used in the contract. The variable block is a set of variables. The timestamp does not always reflect the current time and may be inaccurate. The value of a block can be influenced by miners. Maximal Extractable Value attacks require a timestamp of up to 900 seconds. There is no guarantee that the value is right, all that is guaranteed is that it is higher than the timestamp of the previous block.

Remediation
You can use an Oracle to get the exact time or verify if a delay of 900 seconds won’t impact the logic of the smart contract.

Acknowledged
The Team has acknowledged the risk

Low severity issues

B.3 Floating pragma

```
Line 2:
pragma solidity ^0.8.4;
```

Description
The contract makes use of the floating-point pragma 0.8.4 Contracts should be deployed using the same compiler version and flags that were used during the testing process. Locking the pragma helps ensure that contracts are not unintentionally deployed using another pragma, such as an obsolete version that may introduce issues in the contract system.

Remediation
Consider locking the pragma version. It is advised that floating pragma not be used in production. Both truffle-config.js and hardhat.config.js support locking the pragma version.

Acknowledged
The Team has acknowledged the risk

B.4 Missing Address Verification

```
Line 56:
function initialize(bytes memory data, address _adminAddress,
        address _operatorAddress) external initializer {
        __Ownable_init();
        __Pausable_init();
        __ReentrancyGuard_init();
        uint256 _minBetAmount;
        uint256 _startTime;
        uint256 _lockTime;
        uint8 _rewardRate;
        uint8 _treasuryRate;
        address _gameToken;
        bytes32 _questionID;
        (_minBetAmount, _startTime, _lockTime, _rewardRate, _treasuryRate, _gameToken,
_questionID) = abi.decode(
            data,
            (uint256, uint256, uint256, uint8, uint8, address, bytes32)
        );
        adminAddress = _adminAddress;
        operatorAddress = _operatorAddress;
        gameToken = IERC20Upgradeable(_gameToken);
        questionID = _questionID;
        minBetAmount = _minBetAmount;
        question.startTime = _startTime;
        question.lockTime = _lockTime;
        rewardRate = _rewardRate;
        treasuryRate = _treasuryRate;
    }
```

```
Line 168:
function placeBet(bytes memory data, address userAddress) external onlyOwner
whenNotPaused nonReentrant {
        //Option to be A or B
        uint256 amount;
        uint8 option;
        (amount, option) = abi.decode(data, (uint256, uint8));
        require(option == 0 || option == 1, "Invalid Option");
        //betting should be open
        require(_bettable() && !question.isGameEnded, "Betting should be open");
        //amount >= minBetAmount
        require(amount >= minBetAmount, "Bet amount must be greater than
minBetAmount");
        //can only bet once
        require(ledger[userAddress].amount == 0, "only bet once");
```

Description

Certain functions lack a safety check in the address, the address-type argument should include a zero-address test, otherwise, the contract's functionality may become inaccessible or token may be burned inperpetuity.

Remediation
It’s recommended to undertake further validation prior to user-supplied data. The concerns can be resolved by utilizing a whitelist technique or a modifier.

Solved
The Team has solved the issue using a verification of the address with the zero address.

B.5 Missing Value Verification

```
Line 56:
function initialize(bytes memory data, address _adminAddress,
        address _operatorAddress) external initializer {
        __Ownable_init();
        __Pausable_init();
        __ReentrancyGuard_init();
        uint256 _minBetAmount;
        uint256 _startTime;
        uint256 _lockTime;
        uint8 _rewardRate;
        uint8 _treasuryRate;
        address _gameToken;
        bytes32 _questionID;
        (_minBetAmount, _startTime, _lockTime, _rewardRate, _treasuryRate, _gameToken,
_questionID) = abi.decode(
            data,
            (uint256, uint256, uint256, uint8, uint8, address, bytes32)
        );
        adminAddress = _adminAddress;
        operatorAddress = _operatorAddress;
        gameToken = IERC20Upgradeable(_gameToken);
        questionID = _questionID;
        minBetAmount = _minBetAmount;
        question.startTime = _startTime;
        question.lockTime = _lockTime;
        rewardRate = _rewardRate;
        treasuryRate = _treasuryRate;
        }
```

Description
Certain functions lack a safety check in the arguments values, the sum of rewardRate and treasuryRate should be equal to TOTAL_RATE.
Otherwise, it this could cause some errors in the logic of smart contract.

Remediation
Add a require statement that verifies that the sum of values of rewardRate and treasuryRate is equal to TOTAL_RATE.

Acknowledged
The Team has acknowledged the risk

Automated Tests

Slither

Low level calls:

Use of "delegatecall": should be avoided whenever possible.
External code, that is called can change the state of the calling contract and send ether from the caller's balance.
If this is wanted behaviour, use the Solidity library feature if possible.
more
Pos: 211:50:

Low level calls:

Use of "call": should be avoided whenever possible.
It can lead to unexpected behavior if return value is not handled properly.
Please use Direct Calls via specifying the called contract's interface.
more
Pos: 57:27:

Low level calls:

Use of "call": should be avoided whenever possible.
It can lead to unexpected behavior if return value is not handled properly.
Please use Direct Calls via specifying the called contract's interface.
more
Pos: 131:50:

Low level calls:

Use of "call": should be avoided whenever possible.
It can lead to unexpected behavior if return value is not handled properly.
Please use Direct Calls via specifying the called contract's interface.
more
Pos: 83:54:

Gas costs:

Gas requirement of function Factory.upgradeTo 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: 51:4:

Gas costs:

Gas requirement of function Factory.upgradeToAndCall 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: 64:4:

Gas costs:

Gas requirement of function Factory.initialize 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: 26:4:

Gas costs:

Gas requirement of function Factory.setAdmin 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: 52:4:

Constant/View/Pure functions:

INTERNAL ERROR in module Constant/View/Pure functions: Cannot read property 'name' of undefined
Pos: not available

Similar variable names:

NativeMetaTransaction.executeMetaTransaction(address,bytes,bytes32,bytes32,uint8) : Variables have very similar names "sigR" and "sigS". Note: Modifiers are currently not considered by this static analysis.
Pos: 39:44:

Similar variable names:

NativeMetaTransaction.executeMetaTransaction(address,bytes,bytes32,bytes32,uint8) : Variables have very similar names "sigR" and "sigS". Note: Modifiers are currently not considered by this static analysis.
Pos: 39:50:

Similar variable names:

NativeMetaTransaction.executeMetaTransaction(address,bytes,bytes32,bytes32,uint8) : Variables have very similar names "sigR" and "sigV". Note: Modifiers are currently not considered by this static analysis.
Pos: 39:44:

Similar variable names:

NativeMetaTransaction.executeMetaTransaction(address,bytes,bytes32,bytes32,uint8) : Variables have very similar names "sigR" and "sigV". Note: Modifiers are currently not considered by this static analysis.
Pos: 39:56:

Similar variable names:

NativeMetaTransaction.executeMetaTransaction(address,bytes,bytes32,bytes32,uint8) : Variables have very similar names "sigS" and "sigV". Note: Modifiers are currently not considered by this static analysis.
Pos: 39:50:

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: 53:16:

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: 66:26:

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: 135:15:

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: 197:19:

Mythx

< Standard SCAN contracts/Factory.sol Submitted 96 minutes ago ...

Issues

These were the issues detected in the scan. You can use the toggles to filter by severity and display or hide ignored issues.

Severity Low (1) Medium (0) High (0)

Ignored Issues Hidden (0)

ID	Severity	Name	File	Location
SWC-103	Low	A floating pragma is set.	ERC1967UpgradeUpgradeable.sol	L: 2 C: 0

< Standard SCAN contracts/GameQuestion.sol Submitted 101 minutes ago ...

Issues

These were the issues detected in the scan. You can use the toggles to filter by severity and display or hide ignored issues.

Severity Low (1) Medium (0) High (0)

Ignored Issues Hidden (0)

ID	Severity	Name	File	Location
SWC-103	Low	A floating pragma is set.	ERC1967UpgradeUpgradeable.sol	L: 2 C: 0

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..

Functional test

Factory.sol

Function Names	Technical Result	Logical Result	Overall
initialize	PASS	PASS	PASS
setAdmin	PASS	PASS	PASS
setOperator	PASS	PASS	PASS
addQuestionType	PASS	PASS	PASS
deployQuestions	PASS	PASS	PASS
placeTheBet	PASS	PASS	PASS
claimBet	PASS	PASS	PASS
claimGameBets	PASS	PASS	PASS
claimGameTreasury	PASS	PASS	PASS
getGameQuestions	PASS	PASS	PASS
getGameQuestionAddress	PASS	PASS	PASS

GameQuestion.sol

Function Names	Technical Result	Logical Result	Overall
initialize	PASS	PASS	PASS
setAdmin	PASS	PASS	PASS
setOperator	PASS	PASS	PASS
setRewardRate	PASS	PASS	PASS
setTreasuryRate	PASS	PASS	PASS
setMinBetAmount	PASS	PASS	PASS
_calculateRewards	PASS	PASS	PASS
placeBet	PASS	PASS	PASS
executeQuestion	PASS	PASS	PASS
claim	PASS	PASS	PASS
claimTreasury	PASS	PASS	PASS
seeUserInfo	PASS	PASS	PASS
refundable	PASS	PASS	PASS
claimable	PASS	PASS	PASS
pause	PASS	PASS	PASS

Closing Summary

Overall, smart contracts are very well written and adhere to guidelines. Many issues were discovered during the initial audit; the majority of them are fixed by the Yoda Team.

Disclaimer

Quillhash audit is not a security warranty, investment advice, or an endorsement of the **Yoda Contracts**. 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 **Yoda** 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

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com