QuillAudits

Audit Report
June, 2022

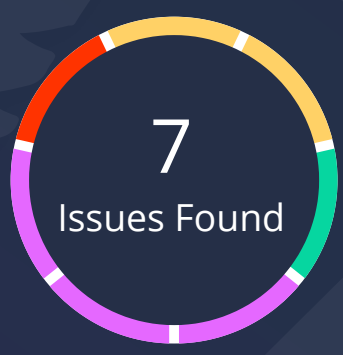
For

THE GAMBLE KINGDOM

Table of Content

Executive Summary

Project Name	The Gamble Kingdom's Staking Platform
Overview	The Gamble Kingdom (TGK) team is building a unique virtual world where poker players can buy and hold our native token $TGK, collect TGK NFTs, participate in game and tournament play, and play-to-earn all within the metaverse kingdom. Players will be able to purchase, earn and collect the Kingdom's native token ($TGK) and use it as currency for tournament, buy-ins, at-game stakes and wagers, as well as other tradeable features within the Kingdom.
Timeline	24 May, 2022 - 14 June, 2022
Method	Manual Review, Functional Testing, Automated Testing etc.
Scope of Audit	The scope of this audit was to analyze The Gamble Kingdom's Staking contract
Code Base	*https://github.com/shrishtieth/TGK/blob/main/contracts/staking.sol*
Commit Hash	26d732ad3c40580fa9987f781426f6487445f9ae0
Fixed In	e0287e56860323a36940beffe33c33fd8005fc48



High
Medium
Low
Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	1	0	0	1
Partially Resolved Issues	0	0	0	0
Resolved Issues	0	2	1	2

Types of Severities

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.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

Checked Vulnerabilities

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- Exception Disorder
- Gasless Send
- Use of tx.origin
- Compiler version not fixed
- Address hardcoded
- Divide before multiply
- Integer overflow/underflow
- Dangerous strict equalities
- Tautology or contradiction
- Return values of low-level calls
- Missing Zero Address Validation
- Private modifier
- Revert/require functions
- Using block.timestamp
- Multiple Sends
- Using SHA3
- Using suicide
- Using throw
- Using inline assembly

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.

Manual Testing

High Severity Issues

1. Incorrect Rewards Calculation

Line #260

```
259
260         _userLastTime[msg.sender] = block.timestamp;
261
```

Description

According to the contract's logic, the users are allowed to make multiple deposts in the contract along with their previous amount and the time at which they last called the deposit function is updated in the contract. That being said, there are a few scenarios in which the rewards will not be accurate. For example;

- Let's suppose the lockup period is of 1 hr .If an user deposits the tokens again after half lockup period is passed and his/her "last time" is updated then in order to withdraw, the user has to wait another hour because of the check in the withdraw function that has a condition of "block.timestamp >= _userLastTime[msg.sender] + lockupPeriod" which states that one can only withdraw after time of last deposit added into lockup period will be passed.
  Thus, the user has to wait for more than the initial lockup period if he/she decides to deposit again.

- _userLastTime is not taken into consideration as it's a check vector in the withdraw function and in case of multiple deposits and the rewards for the previous deposits will reset.

Hence, The lockup period and rewards will be reset of previous deposit amount if an user makes 2 or more deposits

Remediation

It is not recommended to allow users to make multiple deposits and if it is implemented then while calculating rewards, the rewards of previously staked tokens should be kept aside before more tokens are deposited. Therefore, rewards shouldn't reset after a new deposit.

Status

Acknowledged (Intended Behavior)

Medium Severity Issues

2. Missing Imperative Validation

Line #367

```
360     /**
361      * @notice claim all remaining balance on the contract
362      * Residual balance is all the remaining tokens that have not been distributed
363      * (e.g, in case the number of stakeholders is not sufficient)
364      * @dev Can only be called one year after the end of the staking period
365      * Cannot claim initial stakeholders deposit
366      */
        ftrace | funcSig
367     function withdrawResidualBalance() external onlyOwner {
368         uint256 balance = token.balanceOf(address(this));
369         uint256 residualBalance = balance - (_totalStaked);
370         require(residualBalance > 0, "No residual Balance to withdraw");
371         token.safeTransfer(owner(), residualBalance);
372         emit ResidualWithdraw(residualBalance);
373     }
```

Description

As per the annotations by TGK of the withdrawResidueBalance states that it can only be called after an year has passed since the end period of staking but there is no checks in place for such condition. Thus, this function can be called anytime by the owner and doesn't serve the purpose as this is unintended behavior

Remediation

To solve the issue, we would recommend to put a check that will check whether a year after the end period has been passed or not.

Status

Fixed

3. Logic Fail

Line #317

```
317     function withdrawAll() external {
318         require(
319             block.timestamp >= lockupPeriod,
320             "No withdraw until lockup ends"
321         );
```

Description

As per the logic of contract, the withdrawAll() function can only be called once the lockup period is passed, but in this case the lockupPeriod is an uint value in seconds and the check will always pass and this function can be called immediately after deposit. Hence, unintended behavior.

Remediation

To solve the issue, we would recommend to put a check that will check the intended time has been passed since the staking or set the lockupPeriod with respect to block.timestamp rather than a value in seconds.

Status

Fixed

Low Severity Issues

4. Lockup Period not initialized in the constructor

```
90      constructor(
91          address _token1,
92          uint256 _APY1,
93          uint256 _duration1,
94          uint _maxAmountStaked1,
95          address pair1,
96          address oracle1
97      ) {
98          stakingDuration = _duration1;
99          token = IERC20(_token1);
100         APY = _APY1;
101         stakingMax = _maxAmountStaked1;
102         startPeriod = block.timestamp;
103         endPeriod = block.timestamp + stakingDuration;
104         pairContract = IUniswapV2Pair(pair1);
105           ETHPrice = IETHPrice(oracle1);
106         (address token0, ) = (pairContract.token0(), pairContract.token1());
107         _index = _token1 == token0 ? 0 : 1;
108         emit StartStaking(startPeriod,  endPeriod);
```

Description

The value of Lock Up period is not set in the constructor. Although, the contract has a function to update it but it is recommended to initialize it in the constructor because there could be a scenario where after deployment, the owner forgets to update the lockup period and a user deposits, and in that case the staked amount will be available to withdraw right after deposit. Thus, causing unintended behavior.

Remediation

Consider initializing the value of lockup period in the constructor and emit its value in the event as well as it is a crucial parameter for staking.

Status

Fixed

Informational Issues

5. Unnecessary Check

Line #426

```
426     function _calculateRewards(address stakeHolder)
427         public
428         view
429         returns (uint256)
430     {
431         if (startPeriod == 0 || staked[stakeHolder] == 0) {
432             return 0;
433         }
```

Description

As per the logic of contract, the startPeriod will never be equal to zero as it's set in the constructor and there is no need for this.

Remediation

To solve the issue, we would recommend to remove this check or if this is intentional behavior then please do mention in comments with an explanation.

Status

Fixed

6. Unecessary Logic and Operations

Line #447

```
447         bool early = startPeriod > _userStartTime[stakeHolder];
```

Description

As per the logic of contract, the early variable's value will always be false and other conditional operations related to it should be considered redundant.

Remediation

To solve the issue, we would recommend to remove this variable if it's not necessary to contract's intended behavior and if it is, kindly provide an explanation in comments and acknowledge this issue.

Status

Fixed

7. Input validations and error messages can be improved

Description

In the withdraw functions, there are no adequate validations that validates the stakers. Thus, instead of checking whether the caller is a staker in the contract or not, the transaction reverts with the very first error message in the require check. This may result in mishandled events and unclear error messages displayed to the users.

Remediation

It is imperative to include input validations within a function and to display clear error messages to the users in such events.

Status

Acknowledged

Functional Tests (RINKEBY)

- Should update the minimum staking price
- Should update the basic price
- Should update the silver price
- Should update the gold price
- Should update the diamond price
- Should update the ETH Oracle contract
- Should update the APY of staking
- Should update the fee
- Should view the TGK token Price
- Should get latest ETH Price
- Should get TGK Tokens from Price
- Should update the Token Address
- Should update the maximum stake
- Should update the Precision
- Should Deposit the staking amount
- Should call the withdrawal function, check for reward to claim and transfer the amount
- Should call the withdrawAndPayFee function, check for reward to claim and transfer the amount after deducting the fee
- Should call the withdrawAll function, check for reward to claim and withdraw all funds
- Should call the withdrawAllAndPayFees function, check for reward to claim and transfer the amount after deducting the fee
- Should call the withdrawResidualBalance function and transfer all the residual balance into owner's account

Automated Tests

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

Closing Summary

In this report, we have considered the security of The Gamble Kingdom's Staking Platform. We performed our audit according to the procedure described above.

Some issues of Low and informational severity were found, Some suggestions and best practices are also provided in order to improve the code quality and security posture.

In the end,The Gamble Kingdom Team Fixed some issues and Acknowledged other Issues.

Disclaimer

QuillAudits smart contract audit is not a security warranty, investment advice, or an endorsement of The Gamble Kingdom's Staking 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 Gamble Kingdom's Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

500+
Audits Completed

$15B
Secured

500K
Lines of Code Audited

Follow Our Journey

Audit Report
June, 2022

For

THE GAMBLE KINGDOM

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com