QuillAudits

Audit Report
January 2023

For

THE RUG GAME

Table of Content

Executive Summary 01

Checked Vulnerabilities 03

Techniques and Methods 04

Manual Testing 05

A. Contract - STRG.sol 05

B. Contract - THERUGGAME.sol 08

C. Contract - TRG.sol 11

D. Contract - Factory.sol 14

E. Contract - Liquidity.sol 16

F. Common Issues 16

G. Additional issues 19

Functional Testing 22

Automated Testing 22

Closing Summary 28

About QuillAudits 29

Executive Summary

Project Name	TheRugGame
Overview	TheRugGame is a gaming project with a mechanism to rug the token with the least points at the end of the game period.
Timeline	November 5, 2022 - January 4, 2023
Method	Manual Review, Functional Testing, Automated Testing etc.
Scope of Audit	The scope of this audit was to analyze TheRugGame codebase for quality, security, correctness, and exchange listing. *https://github.com/the-games-master/theruggame* “development” branch with commit hash 99d58cbf9c7493a2ef37ad58338de3d371335ff3
Fixed In	6ab4ef4943e4d2e375819787c585ed52c24167bc

High
Medium
Low
Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	1	1	3	2
Partially Resolved Issues	0	0	1	0
Resolved Issues	2	6	3	3

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

A. Contract - STRG.sol

High Severity Issues

No issues found

Medium Severity Issues

1. Centralization issue in rewards and transaction order dependence

Description

The rewards to the stakers of TRG tokens, in the STRG contract are sent in terms of TRG tokens. But the STRG contract does not hold the TRG tokens that are expected to be given for reward. This is because, according to the current logic, the TRG tokens are sent after the stakers have staked some amount of tokens. Only after this, the dividendPerToken variable in TRG contract is updated, thus giving non-zero rewards. But these TRG tokens are sent by the admin, which brings in centralization for providing the rewards to the stakers. If the TRG tokens are not sent to STRG contract(after the stakers have staked), then the stakers will get no rewards.

Also note that currently it is not possible to send TRG tokens to the STRG contract before any stakers come in. This is because the totalStaked is 0, thus giving a panic on line: 42 in TRG contract.

Another issue is that if the TRG tokens are manually sent to the STRG contract each time by the admin for allocating a pool of rewards to the users, an attacker can front run this transaction to deposit a large amount of tokens and then backrun it with withdrawal of his stake, thus sandwiching a substantial amount of rewards, which is unfair to the current stakers(See sandwich attacks).

Remediation

It is advised to allocate TRG rewards in the STRG contract prior to any staking begins to prevent unnecessary/unfair frontrun scenarios and review business and operational logic.

Status

Resolved

Low Severity Issues

No issues found

Informational Issues

2. Arithmetic check

Description

pendingRewards() amount is an unsigned integer member of UserInfo struct and can never be less than zero.

```
if (user.amount <= 0) return 0;                    pendingRewards()
```

The same goes for emergencyWithdraw() function.

```
if (user.amount <= 0) revert NotEnoughDeposit();    emergencyWithdraw()
```

Also in the claimReward() function, userRewards is checked if its less than or equal to zero. But the following less than zero statement is not required as pendingRewards() returns an unsigned integer which can never be less than zero.

```
if (userReward <= 0) revert NotEnoughRewards();     claimReward()
```

Remediation

Consider removing the '<' check as a best practice to avoid any misconceptions or logical errors.

Status

Resolved

3. Staking has no time delay

Description

The staking in STRG contract has no time delay. As a result, there is no incentive for users to stake for a longer time once the rewards in TRG tokens are allocated by the admin to the STRG contract.

Recommendation

In order to incentivize staking for longer time, it is advised to add a time delay for reward claim functionality.

Status

Acknowledged

B. Contract - THERUGGAME.sol

High Severity Issues

No issues found

Medium Severity Issues

1. Insufficient checks in Bribe()

Description

In function bribe the following if statement is incorrect and Insufficient.

```
bool isRugged = IERC20(pair).balanceOf(factory) == 0;
```

This is not a foolproof way to check if a token in the game has been rugged or not. This is because even if the token is Rugged according the flow, still the factory contract can be sent tokens thus setting the isRugged variable to false even if it should have been true.

Recommendation

It is advised to use another way to ensure whether a token has been rugged in the game or not, such as explicitly setting a boolean variable to true or false when the token is rugged.

Status

Acknowledged

Comment: The following issue was resolved by adding a function isValidBribe() in the factory contract which checks if the token was rugged or not.

2. _buyAndBurnToken and _rugLoser susceptible to Frontrun attacks

Description

_buyAndBurnToken() function in susceptible to Frontrun attack. This is because the _amountOutMin parameter is set to 0. This is not recommended as this results in 100% slippage and can be frontrunned by attackers and bots especially in a low liquidity pool.

Similarly _rugLoser() function is susceptible to frontrun attack (see line: 242).

For example, a sandwich attack is possible(which consists of a frontrun and a backrun) to snipe funds from a swap by placing a buy order(with higher gas price) before the user's order and a sell order after the user's order. This would result in the attacker buying at a low price, which results in a costlier trade for the user and then selling at a slightly higher price after the user's trade.

This attack is significant especially in the trades that are larger in size/amount. So here it would mean the transfers of large amounts of THERUGGAME tokens while buying and selling them.

Recommendation

It is advised to set appropriate values for _amountOutMin. Setting it as zero is not recommended as it results in 100% slippage which is most susceptible to frontrun attacks. Review business and operational logic.

Refer

CoinMarketCap
Medium

Additional resource

https://medium.com/mstable/solving-the-issue-with-slippage-in-eip-4626-3af9a5d8e597

Status

Resolved

Comment: The function slippage() was added to set the minimum amount to avoid 100 percent slippage, and is being passed as a parameter in the respective functions.

Low Severity Issues

3. Uncapped supply

Description

There is supply variable that has been used as a parameter in the constructor to mint a specific supply of tokens. But this does not guarantee that there won't be any more tokens minted apart from this supply which can be misleading.

Remediation

It is advised to use an ERC20 Capped to have a capped supply if needed. *https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/extensions/ERC20Capped.sol*

Status

Acknowledged

Informational Issues

4. Naming convention

Description

The afterBalance variable holds the value of the token after the beforeBalance is subtracted from the current balance. This is the balance change, not afterBalance; it is slightly misleading and does not convey the intended message.

```
uint256 afterBalance = IERC20(_token).balanceOf(address(this)) -
    beforeBalance;

uint256 amountToBurn = afterBalance / 2;
uint256 amountForHolders = afterBalance - amountToBurn;
```

Remediation

Consider renaming the variable afterBalance to balanceDifference.

Status

Resolved

C. Contract - TRG.sol

High Severity Issues

1. setsTrg() access control

Description

setsTrg() lacks access control and can be called by anyone to change the sTrg address to any address.

Remediation

Have proper access control set up for the setsTrg function to prevent malicious actors from changing the address to some other contract which can be exploited to deny rewards to users.

Status

Resolved

2. Possibility of Miscalculation of Rewards

Description

If setsTrg() function is not called before the deposit() in the STRG starts, then it will lead to incorrect calculation of pendingRewards and dividendPerToken. By default the sTrg variable be a zero address.

Remediation

Consider making a call to setsTrg() for setting strg mandatory before any deposit activity in the STRG contract begins.

Status

Acknowledged

Medium Severity Issues

3. Usage of Openzeppelin draft-eip contracts is not recommended

Description

Draft EIPs. The following EIPs are still in Draft status. Due to their nature as drafts, the details of these contracts may change and we cannot guarantee their stability. Minor releases of OpenZeppelin Contracts may contain breaking changes for the contracts in this directory, which will be duly announced in the changelog. The EIPs included here are used by projects in production and this may make them less likely to change significantly."

Refer - *https://docs.openzeppelin.com/contracts/4.x/api/token/erc20#ERC20Permit*

Recommendation

Use a more stable version of this code. The ERC20Permit EIP was recently approved for use

Status

Resolved

Low Severity Issues

4. Redundant inheritance

Description

ERC20Permit is not needed to be inherited as ERC20Votes already inherits it.

Remediation

Remove ERC20Permit from being inherited by TheRugGame contract.

Status

Resolved

Informational Issues

5. Unlocked pragma (pragma solidity ^0.8.9)

Description

The solidity pragma version is kept unlocked in STRG.sol

Remediation

It is recommended to lock the pragma solidity version to avoid issues that can be caused with breaking changes in already deployed code.

Status

Resolved

6. Contract naming convention

Description

There are two contracts with similar names in the codebase, TheRugGame and THERUGGAME. It could be more convenient to have the names more explicit and less similar.

Remediation

Consider renaming the affected contracts.

Status

Acknowledged

D. Contract - Factory.sol

High Severity Issues

1. Possible Manipulation of distributeRewardsAndRugLoser()

Description

performUpkeep() is a public function which can be called by anyone but contains a critical function distributeRewardsAndRugLoser(). This can be exploited by anyone to Rug losers quickly in short span of time by directly calling the performUpKeep function, resulting in the game to get over earlier than expected. This could also result in Denial of service and unfair manipulation of the game.

Remediation

Consider reviewing the business and operational logic in order to avoid this issue.

Status

Resolved

Medium Severity Issues

2. Centralization Risk

Description

The onlyOwner functions such as changeCult(), changeDCult(), changeTrg() and emergencyRemoveToken() can be exploited by a malicious admin resulting in a potential Denial of Service attack to users. For example, if the cult and the trg address is changed by a malicious admin to point to some other address, the users will not be able to call the bribe() function because it will lead to execution of the if statement on line: 163 resulting in Denial of Service attack. Additionally emergencyRemoveToken() can be used by a malicious admin to remove tokens that are unfavourable to the winning conditions and can be used to manipulate the game.

Recommendation

It is advised to implement a multi-sig to prevent loss of private keys. Additionally it is advised to set up a governance system to prevent unauthorized changes to state if a compromised contract owner calls the affected functions.

Status

Acknowledged

Low Severity Issues

3. getWinner() can be misleading

Description

getWinner() always shows some winner. This can be misleading. Bcoz this winner may not necessarily be the winner of this round and points can change.

Remediation

It is advised to either change the name of the variable or not allow calling this until it is time to decide the winner.

Status

Resolved

Comment: The function of getting winner and loser was clubbed together. This function now would represent the winner and loser tokens at each point of time according to the points accrued.

E. Contract - Liquidity.sol

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

No issues found

F. Common Issues

High Severity Issues

No issues found

Medium Severity Issues

1. Tokens are pausable

Description

The tokens of TRG, STRG and THERUGGAME can be paused anytime by the owner which could lead to a Denial of Service for users.

Recommendation

Consider removing the pausable functionality if not required to avoid centralization issues. Alternatively governance system can be used in which if a majority consensus is reached then it allows the token to be paused. A blacklist feature can be used to block malicious users from accessing their tokens.

Status

Resolved

Low Severity Issues

2. Ownership transfer

Description

The transferOwnership() function in contract allows the current admin to transfer his privileges to another address. However, inside transferOwnership() , the newOwner is directly stored into the storage owner, after validating the newOwner is a non-zero address, and immediately overwrites the current owner. This can lead to cases where the admin has transferred ownership to an incorrect address and wants to revoke the transfer of ownership or in the cases where the current admin comes to know that the new admin has lost access to his account.

Remediation

It is advised to make ownership transfer a two-step process.
Refer- *https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/Ownable2Step.sol*
and *https://github.com/razzorsec/RazzorSec-Contracts/blob/main/AccessControl/SafeOwn.sol*

Status

Acknowledged

3. Insufficient unit test coverage

Description

The codebase lacks sufficient unit test coverage to test basic functions and whether their working is as expected.

Remediation

It is advised to add sufficient unit tests in the codebase with a test coverage of at least 80 percent. This would also help in better and faster functionality testing and logic verification.

Status

Partially resolved

4. Renounce Ownership

Description

The renounceOwnership function can be called accidentally by the admin leading to immediate renouncement of ownership to zero address after which
any onlyOwner functions will not be callable which can be 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. Refer this post for additional info- *https://www.linkedin.com/posts/razzor_github-razzorsecrazzorsec-contracts-activity-6873251560864968705-HOS8*

Status

Acknowledged

Informational Issues

5. Lack of Comments

Description

The codebase lacks sufficient comments in describing the working and logic of the contracts

Remediation

It is advised to add sufficient comments in the codebase that helps clarify the logic better while auditing.

Status

Acknowledged

G. Additional issues

Note- The following issues were found by the team developer after additional unit test as recommended in the issue F.4

High Severity Issues

No issues found

Medium Severity Issues

1. Incorrect variable added to array

Description

The day was being passed in the array instead of the gameEndTime on line: 346 in the Factory contract. Which meant that for example 31, 32, 33 was being passed instead of 31, 63 and so on. Due to this the rug pull was possible of all the three scenarios of 31, 32 and 33 when 63 was assigned as a value.

Recommendation

It was recommended and fixed by passing gameEndTime into the array by the developer on line: 378

Status

Resolved

2. Funds stuck and transfer to incorrect address

Description

The funds were being sent to the TRG contract instead of the STRG contract. This issue persisted in the function bribe() of THERUGGAME contract and the rugLoser() function of the Factory contract. Due to this funds were being stuck in the TRG contract.

Recommendation

It was recommended and fixed by the team developer to send the funds to correct contract i.e. is the STRG contract on line: 171.

Status

Resolved

3. Incorrect initialization of gameEndTime

Description

The gameEndTime is incorrectly initialized and is expected to be initialized with respect to the gameStartTime assigned. This is because in the checkUpkeep() function, the gameEndTime is being compared to the block.timestamp which won't work if gameEndTime is only initialized in terms of days and not gameStartTime .
Also on the line: 408 the if statement should check that if the block.timestamp is greater than the current gameEndtime (let's say 70 days) but lesser than the validTime (let's say 110). This is bcoz if this is not checked properly, you could end up with the else statement never getting executed. This is because even if the block.timestamp has crossed the validTime, the block.timestamp would still be greater than the _gameEndTime and thus only the else if statement will be triggered and not the else statement.

Recommendation

It is recommended to initialize the _gameEndTime variable correctly and write the if else statements in the checkUpkeep() function accordingly.

Status

Resolved

Low Severity Issues

4. Divide by zero panic

Description

A possible scenario of divide by zero i.e. via the validSupply existed in the codebase on line: 226 in _distributeRewards() function.

Remediation

It was recommended and fixed by the developer to add a check of non-zero value for the same.

Status

Resolved

5. GameStartTime set every time

Description

The gamestartTime was being set in the createToken each time. This was not as per the business logic requirement.

Remediation

According to the new changes in the implementation logic, the setting of the value to gameStartTime was moved to an initialization function instead.

Status

Resolved

Functional Testing

Some of the tests performed are mentioned below:

- ✓ should return token name
- ✓ should not exceed total supply
- ✓ should not mint new tokens
- ✓ should not set sTrg address to a malicious contract
- ✓ should restrict functions when paused
- ✓ should revert with insufficient balances when
- ✓ should restrict functions when paused
- ✓ should be able to bribe with trg or cult token
- ✓ should deposit and withdraw TRG without token loss
- ✓ transfer of THERUGGAME tokens should not affect token accounting
- ✓ should deposit in sTrg contract for multiple addresses
- ✓ should not allocate double rewards for staking
- ✗ should allocate rewards before staking in STRG
- ✗ time delay for staking and claiming rewards

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.

```
ERC1967UpgradeUpgradeable._functionDelegateCall(address,bytes) (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#198-204) uses delegatecall to a input-controlled function id
        - (success,returndata) = target.delegatecall(data) (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#202)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#controlled-delegatecall

Factory._distributeRewards() (contracts/Factory.sol#201-230) ignores return value by IERC20Upgradeable(Liquidity.WETH).transfer(winnerToken,totalActiveReward) (contracts/Factory.sol#213-216)
Factory._rugLoser() (contracts/Factory.sol#232-251) ignores return value by IERC20Upgradeable(trg).transfer(trg,swappedTrg) (contracts/Factory.sol#250)
STRG.deposit(uint256) (contracts/tokens/STRG.sol#35-58) ignores return value by IERC20(trg).transfer(msg.sender,userReward) (contracts/tokens/STRG.sol#44)
STRG.deposit(uint256) (contracts/tokens/STRG.sol#35-58) ignores return value by IERC20(trg).transferFrom(msg.sender,address(this), amount) (contracts/tokens/STRG.sol#48)
STRG.claimReward() (contracts/tokens/STRG.sol#73-82) ignores return value by IERC20(trg).transfer(msg.sender,userReward) (contracts/tokens/STRG.sol#78)
STRG.withdraw(uint256) (contracts/tokens/STRG.sol#84-99) ignores return value by IERC20(trg).transfer(msg.sender, amount + userReward) (contracts/tokens/STRG.sol#90)
STRG.emergencyWithdraw() (contracts/tokens/STRG.sol#101-119) ignores return value by IERC20(trg).transfer(msg.sender,user.amount) (contracts/tokens/STRG.sol#112)
THERUGGAME._buyAndBurnToken(address,uint256) (contracts/tokens/THERUGGAME.sol#113-130) ignores return value by IERC20(_tokenOut).transfer(Liquidity.DEAD_ADDRESS,swappedAmount) (contracts/tokens/THERUGGAME.sol#125)
THERUGGAME.claimReward() (contracts/tokens/THERUGGAME.sol#165-172) ignores return value by IERC20(Liquidity.WETH).transfer(msg.sender,userReward) (contracts/tokens/THERUGGAME.sol#171)
THERUGGAME.bribe(address,uint256) (contracts/tokens/THERUGGAME.sol#174-206) ignores return value by IERC20(_token).transferFrom(msg.sender,address(this), amount) (contracts/tokens/THERUGGAME.sol#187)
THERUGGAME.bribe(address,uint256) (contracts/tokens/THERUGGAME.sol#174-206) ignores return value by IERC20(_token).transfer(Liquidity.DEAD_ADDRESS,amountToBurn) (contracts/tokens/THERUGGAME.sol#197)
THERUGGAME.bribe(address,uint256) (contracts/tokens/THERUGGAME.sol#174-206) ignores return value by IERC20(trg()).transfer(trg(),amountForHolders) (contracts/tokens/THERUGGAME.sol#199)
THERUGGAME.bribe(address,uint256) (contracts/tokens/THERUGGAME.sol#174-206) ignores return value by IERC20(cult()).transfer(dCult(),amountForHolders) (contracts/tokens/THERUGGAME.sol#201)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer

Factory (contracts/Factory.sol#13-401) is an upgradeable contract that does not protect its initiliaze functions: Factory.initialize(address,address,address,uint256,uint256,uint256,uint256) (contracts/Factory.sol#68-102). Anyone
can delete the contract with: UUPSUpgradeable.upgradeTo(address) (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol#72-75)UUPSUpgradeable.upgradeToAndCall(address,bytes) (node_modules/@openzeppelin/
contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol#85-88)Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unprotected-upgradeable-contract

Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        -denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        -inverse = (3 * denominator) ^ 2 (node_modules/@openzeppelin/contracts/utils/math/Math.sol#117)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        -denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        -inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#121)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        -denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        -inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#122)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        -denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        -inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#123)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        -denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        -inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#124)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        -denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        -inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#125)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        -denominator = denominator / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#102)
        -inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#126)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) performs a multiplication on the result of a division:
        -prod0 = prod0 / twos (node_modules/@openzeppelin/contracts/utils/math/Math.sol#105)
        -result = prod0 * inverse (node_modules/@openzeppelin/contracts/utils/math/Math.sol#132)
THERUGGAME._transfer(address,address,uint256) (contracts/tokens/THERUGGAME.sol#56-111) performs a multiplication on the result of a division:
        -burnAmount = (_feesOnContract * burnTax()) / 10000 (contracts/tokens/THERUGGAME.sol#95)
        -points += swappedCult + swappedTrg + (burnAmount * 100000) (contracts/tokens/THERUGGAME.sol#105)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
```

```
THERUGGAME.bribe(address,uint256) (contracts/tokens/THERUGGAME.sol#174-206) uses a dangerous strict equality:
        - isRugged = IERC20(pair).balanceOf(factory) == 0 (contracts/tokens/THERUGGAME.sol#177)
ERC20Votes._writeCheckpoint(ERC20Votes.Checkpoint[],function(uint256,uint256) returns(uint256),uint256) (node_modules/@openzeppelin/contracts/token/ERC20/extensions/ERC20Votes.sol#239-256) uses a dangerous strict equality:
        - pos > 0 && oldCkpt.fromBlock == block.number (node_modules/@openzeppelin/contracts/token/ERC20/extensions/ERC20Votes.sol#251)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities

Reentrancy in Factory._distributeRewards() (contracts/Factory.sol#201-230):
        External calls:
        - IERC20Upgradeable(Liquidity.WETH).transfer(winnerToken,totalActiveReward) (contracts/Factory.sol#213-216)
        State variables written after the call(s):
        - _winnerTotalRewards += totalActiveReward (contracts/Factory.sol#225)
Reentrancy in Factory._rugLoser() (contracts/Factory.sol#232-251):
        External calls:
        - (amountB) = Liquidity.removeLiquidity(loserToken,Liquidity.WETH,address(this)) (contracts/Factory.sol#234-238)
        State variables written after the call(s):
        - delete activeTokens[index] (contracts/Factory.sol#241)
Reentrancy in THERUGGAME._transfer(address,address,uint256) (contracts/tokens/THERUGGAME.sol#56-111):
        External calls:
        - swappedCult = _buyAndBurnToken(cult(),(_feesOnContract * cultTax()) / 10000) (contracts/tokens/THERUGGAME.sol#78-81)
                - IERC20(_tokenIn).approve(ROUTER,_amountIn) (contracts/Liquidity.sol#79)
                - swappedAmount = Liquidity.swap(address(this),_tokenOut,_amountIn,0,address(this)) (contracts/tokens/THERUGGAME.sol#118-124)
                - IERC20(_tokenOut).transfer(Liquidity.DEAD_ADDRESS,swappedAmount) (contracts/tokens/THERUGGAME.sol#125)
                - IUniswapV2Router(ROUTER).swapExactTokensForTokensSupportingFeeOnTransferTokens(_amountIn,_amountOutMin,path,_to,block.timestamp) (contracts/Liquidity.sol#94-101)
        - swappedTrg = _buyAndBurnToken(trg(),(_feesOnContract * trgTax()) / 10000) (contracts/tokens/THERUGGAME.sol#82-85)
                - IERC20(_tokenIn).approve(ROUTER,_amountIn) (contracts/Liquidity.sol#79)
                - swappedAmount = Liquidity.swap(address(this),_tokenOut,_amountIn,0,address(this)) (contracts/tokens/THERUGGAME.sol#118-124)
                - IERC20(_tokenOut).transfer(Liquidity.DEAD_ADDRESS,swappedAmount) (contracts/tokens/THERUGGAME.sol#125)
                - IUniswapV2Router(ROUTER).swapExactTokensForTokensSupportingFeeOnTransferTokens(_amountIn,_amountOutMin,path,_to,block.timestamp) (contracts/Liquidity.sol#94-101)
        - swappedWeth = Liquidity.swap(address(this),Liquidity.WETH,(_feesOnContract * rewardTax()) / 10000,0,factory) (contracts/tokens/THERUGGAME.sol#88-94)
        State variables written after the call(s):
        - super._transfer(address(this),Liquidity.DEAD_ADDRESS,burnAmount) (contracts/tokens/THERUGGAME.sol#96-100)
                - _balances[from] = fromBalance - amount (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#239)
                - _balances[to] += amount (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#242)
        - super._transfer(from,to,amount - feeAmount) (contracts/tokens/THERUGGAME.sol#109)
                - _balances[from] = fromBalance - amount (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#239)
                - _balances[to] += amount (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#242)
        - _feesOnContract = 0 (contracts/tokens/THERUGGAME.sol#106)
Reentrancy in STRG.claimReward() (contracts/tokens/STRG.sol#73-82):
        External calls:
        - IERC20(trg).transfer(msg.sender,userReward) (contracts/tokens/STRG.sol#78)
        State variables written after the call(s):
        - _xDividendPerToken[msg.sender] = dividendPerToken() (contracts/tokens/STRG.sol#81)
        - user.rewardDebt += userReward (contracts/tokens/STRG.sol#80)
Reentrancy in STRG.deposit(uint256) (contracts/tokens/STRG.sol#35-58):
        External calls:
        - IERC20(trg).transfer(msg.sender,userReward) (contracts/tokens/STRG.sol#44)
        State variables written after the call(s):
        - user.rewardDebt += userReward (contracts/tokens/STRG.sol#45)
Reentrancy in STRG.deposit(uint256) (contracts/tokens/STRG.sol#35-58):
        External calls:
        - IERC20(trg).transfer(msg.sender,userReward) (contracts/tokens/STRG.sol#44)
        - IERC20(trg).transferFrom(msg.sender,address(this),_amount) (contracts/tokens/STRG.sol#48)
        State variables written after the call(s):
        - _xDividendPerToken[msg.sender] = dividendPerToken() (contracts/tokens/STRG.sol#55)
        - user.amount += _amount (contracts/tokens/STRG.sol#53)
```

```
Reentrancy in Factory.distributeRewardsAndRugLoser() (contracts/Factory.sol#185-199):
        External calls:
        - _distributeRewards() (contracts/Factory.sol#197)
                - IERC20Upgradeable(Liquidity.WETH).transfer(winnerToken,totalActiveReward) (contracts/Factory.sol#213-216)
        - _rugLoser() (contracts/Factory.sol#198)
                - IERC20(_tokenIn).approve(ROUTER,_amountIn) (contracts/Liquidity.sol#79)
                - IERC20(pair).approve(ROUTER,liquidity) (contracts/Liquidity.sol#59)
                - (amountA,amountB) = IUniswapV2Router(ROUTER).removeLiquidity(_tokenA,_tokenB,liquidity,0,0,_to,block.timestamp) (contracts/Liquidity.sol#61-69)
                - (amountB) = Liquidity.removeLiquidity(loserToken,Liquidity.WETH,address(this)) (contracts/Factory.sol#234-238)
                - swappedTrg = Liquidity.swap(Liquidity.WETH,trg,amountB,0,address(this)) (contracts/Factory.sol#243-249)
                - IERC20Upgradeable(trg).transfer(trg,swappedTrg) (contracts/Factory.sol#250)
                - IUniswapV2Router(ROUTER).swapExactTokensForTokensSupportingFeeOnTransferTokens(_amountIn,_amountOutMin,path,_to,block.timestamp) (contracts/Liquidity.sol#94-101)
        State variables written after the call(s):
        - _rugLoser() (contracts/Factory.sol#198)
                - delete activeTokens[index] (contracts/Factory.sol#241)
        - _rugLoser() (contracts/Factory.sol#198)
                - eliminatedTokens.push(loserToken) (contracts/Factory.sol#240)
Reentrancy in STRG.emergencyWithdraw() (contracts/tokens/STRG.sol#101-119):
        External calls:
        - IERC20(trg).transfer(msg.sender,user.amount) (contracts/tokens/STRG.sol#112)
        State variables written after the call(s):
        - user.amount = 0 (contracts/tokens/STRG.sol#117)
        - user.rewardDebt = 0 (contracts/tokens/STRG.sol#118)
Reentrancy in STRG.withdraw(uint256) (contracts/tokens/STRG.sol#84-99):
        External calls:
        - IERC20(trg).transfer(msg.sender,_amount + userReward) (contracts/tokens/STRG.sol#90)
        State variables written after the call(s):
        - _xDividendPerToken[msg.sender] = dividendPerToken() (contracts/tokens/STRG.sol#96)
        - user.amount -= _amount (contracts/tokens/STRG.sol#93)
        - user.rewardDebt += userReward (contracts/tokens/STRG.sol#94)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1

ERC20Votes._moveVotingPower(address,address,uint256).oldWeight_scope_0 (node_modules/@openzeppelin/contracts/token/ERC20/extensions/ERC20Votes.sol#233) is a local variable never initialized
ERC1967UpgradeUpgradeable._upgradeToAndCallUUPS(address,bytes,bool).slot (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#98) is a local variable never initialized
ERC20Votes._moveVotingPower(address,address,uint256).newWeight_scope_1 (node_modules/@openzeppelin/contracts/token/ERC20/extensions/ERC20Votes.sol#233) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

ERC1967UpgradeUpgradeable._upgradeToAndCallUUPS(address,bytes,bool) (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#87-105) ignores return value by IERC1822ProxiableUpgradeable(newImp
lementation).proxiableUUID() (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#98-102)
Factory.createToken(string,string,uint256,uint256) (contracts/Factory.sol#104-127) ignores return value by Liquidity.addLiquidity(_token,Liquidity.WETH,amountToken,amountWETH,address(this)) (contracts/Factory.sol#116-122)
Factory.requestRandomness(uint32,uint16,uint32) (contracts/Factory.sol#327-338) ignores return value by LINK.transferAndCall(address(VRF_V2_WRAPPER),VRF_V2_WRAPPER.calculateRequestPrice(_callbackGasLimit),abi.encode(_callbackGasL
imit,_requestConfirmations,_numWords)) (contracts/Factory.sol#332-336)
Liquidity.addLiquidity(address,address,uint256,uint256,address) (contracts/Liquidity.sol#15-49) ignores return value by IERC20(_tokenA).approve(ROUTER,_amountA) (contracts/Liquidity.sol#30)
Liquidity.addLiquidity(address,address,uint256,uint256,address) (contracts/Liquidity.sol#15-49) ignores return value by IERC20(_tokenB).approve(ROUTER,_amountB) (contracts/Liquidity.sol#31)
Liquidity.removeLiquidity(address,address,address) (contracts/Liquidity.sol#51-70) ignores return value by IERC20(pair).approve(ROUTER,liquidity) (contracts/Liquidity.sol#59)
Liquidity.swap(address,address,uint256,uint256,address) (contracts/Liquidity.sol#72-105) ignores return value by IERC20(_tokenIn).approve(ROUTER,_amountIn) (contracts/Liquidity.sol#79)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
```

```
ERC20Permit.constructor(string).name (node_modules/@openzeppelin/contracts/token/ERC20/extensions/draft-ERC20Permit.sol#44) shadows:
        - ERC20.name() (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#62-64) (function)
        - IERC20Metadata.name() (node_modules/@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol#17) (function)
THERUGGAME.constructor(string,string,uint256,address)._name (contracts/tokens/THERUGGAME.sol#38) shadows:
        - ERC20._name (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#42) (state variable)
THERUGGAME.constructor(string,string,uint256,address)._symbol (contracts/tokens/THERUGGAME.sol#39) shadows:
        - ERC20._symbol (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#43) (state variable)
MockToken.constructor(string,string)._name (contracts/tokens/mock/MockToken.sol#8) shadows:
        - ERC20._name (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#42) (state variable)
MockToken.constructor(string,string)._symbol (contracts/tokens/mock/MockToken.sol#8) shadows:
        - ERC20._symbol (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#43) (state variable)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

Factory.updateVrfConfiguration(uint32,uint16,uint32,address,address) (contracts/Factory.sol#313-325) should emit an event for:
        - numWords = _numWords (contracts/Factory.sol#322)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic

Factory.updateVrfConfiguration(uint32,uint16,uint32,address,address)._linkAddress (contracts/Factory.sol#317) lacks a zero-check on :
                - linkAddress = _linkAddress (contracts/Factory.sol#323)
Factory.updateVrfConfiguration(uint32,uint16,uint32,address,address)._wrapperAddress (contracts/Factory.sol#318) lacks a zero-check on :
                - wrapperAddress = _wrapperAddress (contracts/Factory.sol#324)
STRG.constructor(address)._trg (contracts/tokens/STRG.sol#31) lacks a zero-check on :
                - trg = _trg (contracts/tokens/STRG.sol#32)
THERUGGAME.constructor(string,string,uint256,address)._factory (contracts/tokens/THERUGGAME.sol#41) lacks a zero-check on :
                - factory = _factory (contracts/tokens/THERUGGAME.sol#44)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Factory._getPoints(address) (contracts/Factory.sol#129-131) has external calls inside a loop: THERUGGAME(_token).points() (contracts/Factory.sol#130)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop

Variable 'ERC1967UpgradeUpgradeable._upgradeToAndCallUUPS(address,bytes,bool).slot (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#98)' in ERC1967UpgradeUpgradeable._upgradeToAndCallU
UPS(address,bytes,bool) (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#87-105) potentially used before declaration: require(bool,string)(slot == _IMPLEMENTATION_SLOT,ERC1967Upgrade:
unsupported proxiableUUID) (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#99)
Variable 'ERC20Votes._moveVotingPower(address,address,uint256).newWeight (node_modules/@openzeppelin/contracts/token/ERC20/extensions/ERC20Votes.sol#228)' in ERC20Votes._moveVotingPower(address,address,uint256) (node_modules/@ope
nzeppelin/contracts/token/ERC20/extensions/ERC20Votes.sol#221-237) potentially used before declaration: (oldWeight,newWeight) = _writeCheckpoint(_checkpoints[dst],_add,amount) (node_modules/@openzeppelin/contracts/token/ERC20/ext
ensions/ERC20Votes.sol#233)
Variable 'ERC20Votes._moveVotingPower(address,address,uint256).oldWeight (node_modules/@openzeppelin/contracts/token/ERC20/extensions/ERC20Votes.sol#228)' in ERC20Votes._moveVotingPower(address,address,uint256) (node_modules/@ope
nzeppelin/contracts/token/ERC20/extensions/ERC20Votes.sol#221-237) potentially used before declaration: (oldWeight,newWeight) = _writeCheckpoint(_checkpoints[dst],_add,amount) (node_modules/@openzeppelin/contracts/token/ERC20/ext
ensions/ERC20Votes.sol#233)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#pre-declaration-usage-of-local-variables

Reentrancy in Factory._distributeRewards() (contracts/Factory.sol#201-230):
        External calls:
        - IERC20Upgradeable(Liquidity.WETH).transfer(winnerToken,totalActiveReward) (contracts/Factory.sol#213-216)
        State variables written after the call(s):
        - dividendPerToken[winnerToken] += (totalActiveReward * 1e18) / validSupply (contracts/Factory.sol#227-229)
        - winnerTotalRewards[winnerToken] += totalActiveReward (contracts/Factory.sol#226)
Reentrancy in Factory._rugLoser() (contracts/Factory.sol#232-251):
        External calls:
        - (amountB) = Liquidity.removeLiquidity(loserToken,Liquidity.WETH,address(this)) (contracts/Factory.sol#234-238)
        State variables written after the call(s):
        - eliminatedTokens.push(loserToken) (contracts/Factory.sol#240)
```

```
Reentrancy in THERUGGAME._transfer(address,address,uint256) (contracts/tokens/THERUGGAME.sol#56-111):
        External calls:
        - swappedCult = _buyAndBurnToken(cult(),(_feesOnContract * cultTax()) / 10000) (contracts/tokens/THERUGGAME.sol#78-81)
                - IERC20(_tokenIn).approve(ROUTER,_amountIn) (contracts/Liquidity.sol#79)
                - swappedAmount = Liquidity.swap(address(this),_tokenOut,_amountIn,0,address(this)) (contracts/tokens/THERUGGAME.sol#118-124)
                - IERC20(_tokenOut).transfer(Liquidity.DEAD_ADDRESS,swappedAmount) (contracts/tokens/THERUGGAME.sol#125)
                - IUniswapV2Router(ROUTER).swapExactTokensForTokensSupportingFeeOnTransferTokens(_amountIn,_amountOutMin,path,_to,block.timestamp) (contracts/Liquidity.sol#94-101)
        - swappedTrg = _buyAndBurnToken(trg(),(_feesOnContract * trgTax()) / 10000) (contracts/tokens/THERUGGAME.sol#82-85)
                - IERC20(_tokenIn).approve(ROUTER,_amountIn) (contracts/Liquidity.sol#79)
                - swappedAmount = Liquidity.swap(address(this),_tokenOut,_amountIn,0,address(this)) (contracts/tokens/THERUGGAME.sol#118-124)
                - IERC20(_tokenOut).transfer(Liquidity.DEAD_ADDRESS,swappedAmount) (contracts/tokens/THERUGGAME.sol#125)
                - IUniswapV2Router(ROUTER).swapExactTokensForTokensSupportingFeeOnTransferTokens(_amountIn,_amountOutMin,path,_to,block.timestamp) (contracts/Liquidity.sol#94-101)
        - swappedWeth = Liquidity.swap(address(this),Liquidity.WETH,(_feesOnContract * rewardTax()) / 10000,0,factory) (contracts/tokens/THERUGGAME.sol#88-94)
        State variables written after the call(s):
        - points += swappedCult + swappedTrg + (burnAmount * 100000) (contracts/tokens/THERUGGAME.sol#105)
        - wethReward += swappedWeth (contracts/tokens/THERUGGAME.sol#102)
Reentrancy in THERUGGAME.bribe(address,uint256) (contracts/tokens/THERUGGAME.sol#174-206):
        External calls:
        - IERC20(_token).transferFrom(msg.sender,address(this),_amount) (contracts/tokens/THERUGGAME.sol#187)
        - IERC20(_token).transfer(Liquidity.DEAD_ADDRESS,amountToBurn) (contracts/tokens/THERUGGAME.sol#197)
        - IERC20(trg()).transfer(trg(),amountForHolders) (contracts/tokens/THERUGGAME.sol#199)
        - IERC20(cult()).transfer(dCult(),amountForHolders) (contracts/tokens/THERUGGAME.sol#201)
        State variables written after the call(s):
        - points += (_amount * 3) / 2 (contracts/tokens/THERUGGAME.sol#203)
Reentrancy in Factory.createToken(string,string,uint256,uint256) (contracts/Factory.sol#104-127):
        External calls:
        - Liquidity.addLiquidity(_token,Liquidity.WETH,amountToken,amountWETH,address(this)) (contracts/Factory.sol#116-122)
        State variables written after the call(s):
        - gameStartTime = block.timestamp (contracts/Factory.sol#124)
Reentrancy in STRG.deposit(uint256) (contracts/tokens/STRG.sol#35-58):
        External calls:
        - IERC20(trg).transfer(msg.sender,userReward) (contracts/tokens/STRG.sol#44)
        - IERC20(trg).transferFrom(msg.sender,address(this),_amount) (contracts/tokens/STRG.sol#48)
        State variables written after the call(s):
        - _mint(msg.sender,_amount) (contracts/tokens/STRG.sol#51)
                - _balances[account] += amount (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#267)
        - _mint(msg.sender,_amount) (contracts/tokens/STRG.sol#51)
                - _totalSupply += amount (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#264)
        - totalStaked += _amount (contracts/tokens/STRG.sol#54)
Reentrancy in STRG.emergencyWithdraw() (contracts/tokens/STRG.sol#101-119):
        External calls:
        - IERC20(trg).transfer(msg.sender,user.amount) (contracts/tokens/STRG.sol#112)
        State variables written after the call(s):
        - _burn(msg.sender,user.amount) (contracts/tokens/STRG.sol#113)
                - _balances[account] = accountBalance - amount (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#293)
        - _burn(msg.sender,user.amount) (contracts/tokens/STRG.sol#113)
                - _totalSupply -= amount (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#295)
Reentrancy in STRG.withdraw(uint256) (contracts/tokens/STRG.sol#84-99):
        External calls:
        - IERC20(trg).transfer(msg.sender,_amount + userReward) (contracts/tokens/STRG.sol#90)
        State variables written after the call(s):
        - _burn(msg.sender,_amount) (contracts/tokens/STRG.sol#91)
                - _balances[account] = accountBalance - amount (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#293)
        - _burn(msg.sender,_amount) (contracts/tokens/STRG.sol#91)
                - _totalSupply -= amount (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#295)
        - totalStaked -= _amount (contracts/tokens/STRG.sol#95)
```

```
Reentrancy in THERUGGAME._transfer(address,address,uint256) (contracts/tokens/THERUGGAME.sol#56-111):
        External calls:
        - swappedCult = _buyAndBurnToken(cult(),(_feesOnContract * cultTax()) / 10000) (contracts/tokens/THERUGGAME.sol#78-81)
                - IERC20(_tokenIn).approve(ROUTER,_amountIn) (contracts/Liquidity.sol#79)
                - swappedAmount = Liquidity.swap(address(this),_tokenOut,_amountIn,0,address(this)) (contracts/tokens/THERUGGAME.sol#118-124)
                - IERC20(_tokenOut).transfer(Liquidity.DEAD_ADDRESS,swappedAmount) (contracts/tokens/THERUGGAME.sol#125)
                - IUniswapV2Router(ROUTER).swapExactTokensForTokensSupportingFeeOnTransferTokens(_amountIn, amountOutMin,path,_to,block.timestamp) (contracts/Liquidity.sol#94-101)
        - swappedTrg = _buyAndBurnToken(trg(),(_feesOnContract * trgTax()) / 10000) (contracts/tokens/THERUGGAME.sol#82-85)
                - IERC20(_tokenIn).approve(ROUTER,_amountIn) (contracts/Liquidity.sol#79)
                - swappedAmount = Liquidity.swap(address(this),_tokenOut,_amountIn,0,address(this)) (contracts/tokens/THERUGGAME.sol#118-124)
                - IERC20(_tokenOut).transfer(Liquidity.DEAD_ADDRESS,swappedAmount) (contracts/tokens/THERUGGAME.sol#125)
                - IUniswapV2Router(ROUTER).swapExactTokensForTokensSupportingFeeOnTransferTokens(_amountIn, amountOutMin,path,_to,block.timestamp) (contracts/Liquidity.sol#94-101)
        - swappedWeth = Liquidity.swap(address(this),Liquidity.WETH,(_feesOnContract * rewardTax()) / 10000,0,factory) (contracts/tokens/THERUGGAME.sol#88-94)
        Event emitted after the call(s):
        - Transfer(from,to,amount) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#245)
                - super._transfer(address(this),Liquidity.DEAD_ADDRESS,burnAmount) (contracts/tokens/THERUGGAME.sol#96-100)
        - Transfer(from,to,amount) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#245)
                - super._transfer(from,to,amount - feeAmount) (contracts/tokens/THERUGGAME.sol#109)
Reentrancy in THERUGGAME.bribe(address,uint256) (contracts/tokens/THERUGGAME.sol#174-206):
        External calls:
        - IERC20(_token).transferFrom(msg.sender,address(this),_amount) (contracts/tokens/THERUGGAME.sol#187)
        - IERC20(_token).transfer(Liquidity.DEAD_ADDRESS,amountToBurn) (contracts/tokens/THERUGGAME.sol#197)
        - IERC20(trg()).transfer(trg(),amountForHolders) (contracts/tokens/THERUGGAME.sol#199)
        - IERC20(cult()).transfer(dCult(),amountForHolders) (contracts/tokens/THERUGGAME.sol#201)
        Event emitted after the call(s):
        - Bribe(_token,address(this),_amount) (contracts/tokens/THERUGGAME.sol#205)
Reentrancy in Factory.createToken(string,string,uint256,uint256) (contracts/Factory.sol#104-127):
        External calls:
        - Liquidity.addLiquidity(_token,Liquidity.WETH,amountToken,amountWETH,address(this)) (contracts/Factory.sol#116-122)
        Event emitted after the call(s):
        - TokenCreated(_token) (contracts/Factory.sol#126)
Reentrancy in STRG.deposit(uint256) (contracts/tokens/STRG.sol#35-58):
        External calls:
        - IERC20(trg).transfer(msg.sender,userReward) (contracts/tokens/STRG.sol#44)
        - IERC20(trg).transferFrom(msg.sender,address(this),_amount) (contracts/tokens/STRG.sol#48)
        Event emitted after the call(s):
        - Deposit(msg.sender,_amount) (contracts/tokens/STRG.sol#57)
        - Transfer(address(0),account,amount) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#269)
                - _mint(msg.sender,_amount) (contracts/tokens/STRG.sol#51)
Reentrancy in STRG.emergencyWithdraw() (contracts/tokens/STRG.sol#101-119):
        External calls:
        - IERC20(trg).transfer(msg.sender,user.amount) (contracts/tokens/STRG.sol#112)
        Event emitted after the call(s):
        - EmergencyWithdraw(msg.sender,user.amount) (contracts/tokens/STRG.sol#115)
        - Transfer(account,address(0),amount) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#298)
                - _burn(msg.sender,user.amount) (contracts/tokens/STRG.sol#113)
Reentrancy in STRG.withdraw(uint256) (contracts/tokens/STRG.sol#84-99):
        External calls:
        - IERC20(trg).transfer(msg.sender,_amount + userReward) (contracts/tokens/STRG.sol#90)
        Event emitted after the call(s):
        - Transfer(account,address(0),amount) (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#298)
                - _burn(msg.sender,_amount) (contracts/tokens/STRG.sol#91)
        - Withdraw(msg.sender,_amount) (contracts/tokens/STRG.sol#98)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
```

```
ERC20Votes.delegateBySig(address,uint256,uint256,uint8,bytes32,bytes32) (node_modules/@openzeppelin/contracts/token/ERC20/extensions/ERC20Votes.sol#146-163) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp <= expiry,ERC20Votes: signature expired) (node_modules/@openzeppelin/contracts/token/ERC20/extensions/ERC20Votes.sol#154)
ERC20Permit.permit(address,address,uint256,uint256,uint8,bytes32,bytes32) (node_modules/@openzeppelin/contracts/token/ERC20/extensions/draft-ERC20Permit.sol#49-68) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp <= deadline,ERC20Permit: expired deadline) (node_modules/@openzeppelin/contracts/token/ERC20/extensions/draft-ERC20Permit.sol#58)
Factory.distributeRewardsAndRugLoser() (contracts/Factory.sol#185-199) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp < validTime (contracts/Factory.sol#190)
Factory.checkUpkeep(bytes) (contracts/Factory.sol#370-393) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp > _gameEndTime (contracts/Factory.sol#385)
        - upkeepNeeded = block.timestamp >= validTime (contracts/Factory.sol#391)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

AddressUpgradeable._revert(bytes,string) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#206-218) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#211-214)
StorageSlotUpgradeable.getAddressSlot(bytes32) (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#52-57) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#54-56)
StorageSlotUpgradeable.getBooleanSlot(bytes32) (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#62-67) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#64-66)
StorageSlotUpgradeable.getBytes32Slot(bytes32) (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#72-77) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#74-76)
StorageSlotUpgradeable.getUint256Slot(bytes32) (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#82-87) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#84-86)
ERC20Votes._unsafeAccess(ERC20Votes.Checkpoint[],uint256) (node_modules/@openzeppelin/contracts/token/ERC20/extensions/ERC20Votes.sol#266-271) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/token/ERC20/extensions/ERC20Votes.sol#267-270)
Strings.toString(uint256) (node_modules/@openzeppelin/contracts/utils/Strings.sol#18-38) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/Strings.sol#24-26)
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/Strings.sol#30-32)
ECDSA.tryRecover(bytes32,bytes) (node_modules/@openzeppelin/contracts/utils/cryptography/ECDSA.sol#55-72) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/cryptography/ECDSA.sol#63-67)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/math/Math.sol#66-70)
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/math/Math.sol#86-93)
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/math/Math.sol#100-109)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Different versions of Solidity are used:
        - Version used: ['=0.8.9', '^0.8.0', '^0.8.1', '^0.8.2', '^0.8.9']
        - ^0.8.0 (node_modules/@chainlink/contracts/src/v0.8/interfaces/AutomationCompatibleInterface.sol#2)
        - ^0.8.0 (node_modules/@chainlink/contracts/src/v0.8/interfaces/LinkTokenInterface.sol#2)
        - ^0.8.0 (node_modules/@chainlink/contracts/src/v0.8/interfaces/VRFV2WrapperInterface.sol#2)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/interfaces/draft-IERC1822Upgradeable.sol#4)
        - ^0.8.2 (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/proxy/beacon/IBeaconUpgradeable.sol#4)
        - ^0.8.2 (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/IERC20Upgradeable.sol#4)
        - ^0.8.1 (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/governance/utils/IVotes.sol#3)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/security/Pausable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#4)
```

```
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/extensions/ERC20Burnable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/extensions/ERC20Votes.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/extensions/draft-ERC20Permit.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/extensions/draft-IERC20Permit.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Counters.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Strings.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/cryptography/ECDSA.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/cryptography/EIP712.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/math/Math.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/math/SafeCast.sol#5)
        - =0.8.9 (contracts/Factory.sol#2)
        - =0.8.9 (contracts/Liquidity.sol#2)
        - =0.8.9 (contracts/interfaces/IFactory.sol#2)
        - =0.8.9 (contracts/interfaces/IStrg.sol#2)
        - =0.8.9 (contracts/interfaces/ITrg.sol#2)
        - =0.8.9 (contracts/interfaces/IUniswap.sol#2)
        - ^0.8.9 (contracts/tokens/STRG.sol#2)
        - =0.8.9 (contracts/tokens/THERUGGAME.sol#2)
        - =0.8.9 (contracts/tokens/TRG.sol#2)
        - =0.8.9 (contracts/tokens/mock/MockToken.sol#2)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used

TheRugGame._burn(address,uint256) (contracts/tokens/TRG.sol#65-70) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.0 (node_modules/@chainlink/contracts/src/v0.8/interfaces/AutomationCompatibleInterface.sol#2) allows old versions
Pragma version^0.8.0 (node_modules/@chainlink/contracts/src/v0.8/interfaces/LinkTokenInterface.sol#2) allows old versions
Pragma version^0.8.0 (node_modules/@chainlink/contracts/src/v0.8/interfaces/VRFV2WrapperInterface.sol#2) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/interfaces/draft-IERC1822Upgradeable.sol#4) allows old versions
Pragma version^0.8.2 (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/proxy/beacon/IBeaconUpgradeable.sol#4) allows old versions
Pragma version^0.8.2 (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/IERC20Upgradeable.sol#4) allows old versions
Pragma version^0.8.1 (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts-upgradeable/utils/StorageSlotUpgradeable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/governance/utils/IVotes.sol#3) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/security/Pausable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/extensions/ERC20Burnable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/extensions/ERC20Votes.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/extensions/draft-ERC20Permit.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/extensions/draft-IERC20Permit.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Counters.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Strings.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/cryptography/ECDSA.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/cryptography/EIP712.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/math/Math.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/math/SafeCast.sol#5) allows old versions
Pragma version=0.8.9 (contracts/Factory.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version=0.8.9 (contracts/Liquidity.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version=0.8.9 (contracts/interfaces/IFactory.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version=0.8.9 (contracts/interfaces/IStrg.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version=0.8.9 (contracts/interfaces/ITrg.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version=0.8.9 (contracts/interfaces/IUniswap.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version^0.8.9 (contracts/tokens/STRG.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version=0.8.9 (contracts/tokens/THERUGGAME.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version=0.8.9 (contracts/tokens/TRG.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
Pragma version=0.8.9 (contracts/tokens/mock/MockToken.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
solc-0.8.9 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in ERC1967UpgradeUpgradeable._functionDelegateCall(address,bytes) (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#198-204):
        - (success,returndata) = target.delegatecall(data) (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#202)
Low level call in AddressUpgradeable.sendValue(address,uint256) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#60-65):
        - (success) = recipient.call{value: amount}() (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#63)
Low level call in AddressUpgradeable.functionCallWithValue(address,bytes,uint256,string) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#128-137):
        - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#135)
Low level call in AddressUpgradeable.functionStaticCall(address,bytes,string) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#155-162):
        - (success,returndata) = target.staticcall(data) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#160)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

STRG (contracts/tokens/STRG.sol#10-140) should inherit from ITrg (contracts/interfaces/ITrg.sol#4-6)
THERUGGAME (contracts/tokens/THERUGGAME.sol#17-247) should inherit from ITrg (contracts/interfaces/ITrg.sol#4-6)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-inheritance

Function OwnableUpgradeable.__Ownable_init() (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#29-31) is not in mixedCase
Function OwnableUpgradeable.__Ownable_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#33-35) is not in mixedCase
Variable OwnableUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol#94) is not in mixedCase
Function ERC1967UpgradeUpgradeable.__ERC1967Upgrade_init() (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#21-22) is not in mixedCase
Function ERC1967UpgradeUpgradeable.__ERC1967Upgrade_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#24-25) is not in mixedCase
Variable ERC1967UpgradeUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/proxy/ERC1967/ERC1967UpgradeUpgradeable.sol#211) is not in mixedCase
Function UUPSUpgradeable.__UUPSUpgradeable_init() (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol#23-24) is not in mixedCase
Function UUPSUpgradeable.__UUPSUpgradeable_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol#26-27) is not in mixedCase
Variable UUPSUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol#107) is not in mixedCase
Variable UUPSUpgradeable.__self (node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol#29) is not in mixedCase
Function ContextUpgradeable.__Context_init() (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#18-19) is not in mixedCase
Function ContextUpgradeable.__Context_init_unchained() (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#21-22) is not in mixedCase
Variable ContextUpgradeable.__gap (node_modules/@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol#36) is not in mixedCase
Function ERC20Permit.DOMAIN_SEPARATOR() (node_modules/@openzeppelin/contracts/token/ERC20/extensions/draft-ERC20Permit.sol#81-83) is not in mixedCase
Variable ERC20Permit._PERMIT_TYPEHASH_DEPRECATED_SLOT (node_modules/@openzeppelin/contracts/token/ERC20/extensions/draft-ERC20Permit.sol#37) is not in mixedCase
Function IERC20Permit.DOMAIN_SEPARATOR() (node_modules/@openzeppelin/contracts/token/ERC20/extensions/draft-IERC20Permit.sol#59) is not in mixedCase
Variable EIP712._CACHED_DOMAIN_SEPARATOR (node_modules/@openzeppelin/contracts/utils/cryptography/EIP712.sol#31) is not in mixedCase
Variable EIP712._CACHED_CHAIN_ID (node_modules/@openzeppelin/contracts/utils/cryptography/EIP712.sol#32) is not in mixedCase
Variable EIP712._CACHED_THIS (node_modules/@openzeppelin/contracts/utils/cryptography/EIP712.sol#33) is not in mixedCase
Variable EIP712._HASHED_NAME (node_modules/@openzeppelin/contracts/utils/cryptography/EIP712.sol#35) is not in mixedCase
Variable EIP712._HASHED_VERSION (node_modules/@openzeppelin/contracts/utils/cryptography/EIP712.sol#36) is not in mixedCase
Variable EIP712._TYPE_HASH (node_modules/@openzeppelin/contracts/utils/cryptography/EIP712.sol#37) is not in mixedCase
Parameter Factory.initialize(address,address,address,uint256,uint256,uint256,uint256)._trg (contracts/Factory.sol#69) is not in mixedCase
Parameter Factory.initialize(address,address,address,uint256,uint256,uint256,uint256)._cult (contracts/Factory.sol#70) is not in mixedCase
Parameter Factory.initialize(address,address,address,uint256,uint256,uint256,uint256)._dCult (contracts/Factory.sol#71) is not in mixedCase
Parameter Factory.initialize(address,address,address,uint256,uint256,uint256,uint256)._burnTax (contracts/Factory.sol#72) is not in mixedCase
Parameter Factory.initialize(address,address,address,uint256,uint256,uint256,uint256)._cultTax (contracts/Factory.sol#73) is not in mixedCase
Parameter Factory.initialize(address,address,address,uint256,uint256,uint256,uint256)._rewardTax (contracts/Factory.sol#74) is not in mixedCase
Parameter Factory.initialize(address,address,address,uint256,uint256,uint256,uint256)._trgTax (contracts/Factory.sol#75) is not in mixedCase
```

```
Parameter Factory.changeCult(address)._cult (contracts/Factory.sol#253) is not in mixedCase
Parameter Factory.changeDCult(address)._dCult (contracts/Factory.sol#260) is not in mixedCase
Parameter Factory.changeTrg(address)._trg (contracts/Factory.sol#267) is not in mixedCase
Parameter Factory.changeTaxes(uint256,uint256,uint256,uint256)._burnTax (contracts/Factory.sol#275) is not in mixedCase
Parameter Factory.changeTaxes(uint256,uint256,uint256,uint256)._cultTax (contracts/Factory.sol#276) is not in mixedCase
Parameter Factory.changeTaxes(uint256,uint256,uint256,uint256)._rewardTax (contracts/Factory.sol#277) is not in mixedCase
Parameter Factory.changeTaxes(uint256,uint256,uint256,uint256)._trgTax (contracts/Factory.sol#278) is not in mixedCase
Parameter Factory.emergencyRemoveToken(uint256)._index (contracts/Factory.sol#293) is not in mixedCase
Parameter Factory.pauseToken(uint256)._gameTokenIndex (contracts/Factory.sol#298) is not in mixedCase
Parameter Factory.unpauseToken(uint256)._gameTokenIndex (contracts/Factory.sol#302) is not in mixedCase
Parameter Factory.updateVrfConfiguration(uint32,uint16,uint32,address,address)._callbackGasLimit (contracts/Factory.sol#314) is not in mixedCase
Parameter Factory.updateVrfConfiguration(uint32,uint16,uint32,address,address)._requestConfirmations (contracts/Factory.sol#315) is not in mixedCase
Parameter Factory.updateVrfConfiguration(uint32,uint16,uint32,address,address)._numWords (contracts/Factory.sol#316) is not in mixedCase
Parameter Factory.updateVrfConfiguration(uint32,uint16,uint32,address,address)._linkAddress (contracts/Factory.sol#317) is not in mixedCase
Parameter Factory.updateVrfConfiguration(uint32,uint16,uint32,address,address)._wrapperAddress (contracts/Factory.sol#318) is not in mixedCase
Parameter Factory.requestRandomness(uint32,uint16,uint32)._callbackGasLimit (contracts/Factory.sol#328) is not in mixedCase
Parameter Factory.requestRandomness(uint32,uint16,uint32)._requestConfirmations (contracts/Factory.sol#329) is not in mixedCase
Parameter Factory.requestRandomness(uint32,uint16,uint32)._numWords (contracts/Factory.sol#330) is not in mixedCase
Parameter Factory.fulfillRandomWords(uint256,uint256[])._randomWords (contracts/Factory.sol#348) is not in mixedCase
Parameter Factory.rawFulfillRandomWords(uint256,uint256[])._requestId (contracts/Factory.sol#358) is not in mixedCase
Parameter Factory.rawFulfillRandomWords(uint256,uint256[])._randomWords (contracts/Factory.sol#359) is not in mixedCase
Variable Factory.LINK (contracts/Factory.sol#36) is not in mixedCase
Variable Factory.VRF_V2_WRAPPER (contracts/Factory.sol#37) is not in mixedCase
Parameter Liquidity.addLiquidity(address,address,uint256,uint256,address)._tokenA (contracts/Liquidity.sol#16) is not in mixedCase
Parameter Liquidity.addLiquidity(address,address,uint256,uint256,address)._tokenB (contracts/Liquidity.sol#17) is not in mixedCase
Parameter Liquidity.addLiquidity(address,address,uint256,uint256,address)._amountA (contracts/Liquidity.sol#18) is not in mixedCase
Parameter Liquidity.addLiquidity(address,address,uint256,uint256,address)._amountB (contracts/Liquidity.sol#19) is not in mixedCase
Parameter Liquidity.addLiquidity(address,address,uint256,uint256,address)._to (contracts/Liquidity.sol#20) is not in mixedCase
Parameter Liquidity.removeLiquidity(address,address,address)._tokenA (contracts/Liquidity.sol#52) is not in mixedCase
Parameter Liquidity.removeLiquidity(address,address,address)._tokenB (contracts/Liquidity.sol#53) is not in mixedCase
Parameter Liquidity.removeLiquidity(address,address,address)._to (contracts/Liquidity.sol#54) is not in mixedCase
Parameter Liquidity.swap(address,address,uint256,uint256,address)._tokenIn (contracts/Liquidity.sol#73) is not in mixedCase
Parameter Liquidity.swap(address,address,uint256,uint256,address)._tokenOut (contracts/Liquidity.sol#74) is not in mixedCase
Parameter Liquidity.swap(address,address,uint256,uint256,address)._amountIn (contracts/Liquidity.sol#75) is not in mixedCase
Parameter Liquidity.swap(address,address,uint256,uint256,address)._amountOutMin (contracts/Liquidity.sol#76) is not in mixedCase
Parameter Liquidity.swap(address,address,uint256,uint256,address)._to (contracts/Liquidity.sol#77) is not in mixedCase
Parameter Liquidity.getPair(address,address)._tokenA (contracts/Liquidity.sol#107) is not in mixedCase
Parameter Liquidity.getPair(address,address)._tokenB (contracts/Liquidity.sol#107) is not in mixedCase
Parameter STRG.deposit(uint256)._amount (contracts/tokens/STRG.sol#35) is not in mixedCase
Parameter STRG.pendingRewards(address)._user (contracts/tokens/STRG.sol#60) is not in mixedCase
Parameter STRG.withdraw(uint256)._amount (contracts/tokens/STRG.sol#84) is not in mixedCase
Variable STRG._dividendPerToken (contracts/tokens/STRG.sol#16) is not in mixedCase
Parameter THERUGGAME.pendingRewards(address)._user (contracts/tokens/THERUGGAME.sol#153) is not in mixedCase
Parameter THERUGGAME.bribe(address,uint256)._token (contracts/tokens/THERUGGAME.sol#174) is not in mixedCase
Parameter THERUGGAME.bribe(address,uint256)._amount (contracts/tokens/THERUGGAME.sol#174) is not in mixedCase
Parameter TheRugGame.setsTrg(address)._sTrg (contracts/tokens/TRG.sol#22) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Variable IUniswapV2Router.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amountADesired (contracts/interfaces/IUniswap.sol#8) is too similar to IUniswapV2Router.addLiquidity(address,address,uint256,
uint256,uint256,uint256,address,uint256).amountBDesired (contracts/interfaces/IUniswap.sol#9)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-are-too-similar

Factory.initialize(address,address,address,uint256,uint256,uint256,uint256) (contracts/Factory.sol#68-102) uses literals with too many digits:
        - callbackGasLimit = 300000 (contracts/Factory.sol#94)
Liquidity.slitherConstructorConstantVariables() (contracts/Liquidity.sol#7-114) uses literals with too many digits:
        - DEAD_ADDRESS = 0x000000000000000000000000000000000000dEaD (contracts/Liquidity.sol#12-13)
```

Closing Summary

In this report, we have considered the security of TheRugGame. We performed our audit according to the procedure described above.

Some Issues of high, medium, low and informational severity were found during the course of the audit.

Disclaimer

QuillAudits smart contract audit is not a security warranty, investment advice, or an endorsement of the TheRugGame 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 TheRugGame 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.

700+
Audits Completed

$16B
Secured

700K
Lines of Code Audited

Follow Our Journey

Audit Report
January, 2023

For

THE RUG GAME

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com