Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

# SMART CONTRACT

## Security Audit Report

| | |
|---|---|
| Project: | Remnant Protocol |
| Website: | https://remnant.gg |
| Platform: | Ethereum |
| Language: | Solidity |
| Date: | April 10th, 2022 |

# Table of contents




**Email: audit@EtherAuthority.io**

`

THIS IS SECURITY AUDIT REPORT DOCUMENT AND WHICH MAY CONTAIN INFORMATION WHICH IS CONFIDENTIAL. WHICH INCLUDES ANY POTENTIAL VULNERABILITIES AND MALICIOUS CODES WHICH CAN BE USED TO EXPLOIT THE SOFTWARE. THIS MUST BE REFERRED INTERNALLY AND ONLY SHOULD BE MADE AVAILABLE TO THE PUBLIC AFTER ISSUES ARE RESOLVED.


**Email: audit@EtherAuthority.io**

# Introduction

EtherAuthority was contracted by the Remnant team to perform the Security audit of the Remnant Token and Staking smart contract code. The audit has been performed using manual analysis as well as using automated software tools. This report presents all the findings regarding the audit performed on April 10th, 2022.

**The purpose of this audit was to address the following:**

- Ensure that all claimed functions exist and function correctly.
- Identify any security vulnerabilities that may be present in the smart contract.

# Project Background

Remnant token contract is an ERC20 token standard smart contract which has a “Bot Prevention” mechanism to prevent sniping the liquidity by the bots. Owner can enable or disable the bot prevention mechanism. Remnant staking contract allows users to stake the tokens and let them earn the staking rewards.

The Remnant Protocol contract inherits the IERC20, ERC20, Ownable standard smart contracts from the OpenZeppelin library. These OpenZeppelin contracts are considered community-audited and time-tested, and hence are not part of the audit scope.

# Audit scope

| **Name** | **Code Review and Security Analysis Report for Remnant Protocol Smart Contracts** |
|---|---|
| **Platform** | **Ethereum / Solidity** |
| **File 1 Online Code Link** | https://github.com/Remnant-Labs/public-repo/blob/main/RemnantToken.sol |
| **File 1 Git Commit** | eec2f87b68267e2f1bc35a9d105b139d55e51561 |
| **File 2 Online Code Link** | https://github.com/Remnant-Labs/public-repo/blob/main/StakingVested.sol |
| **File 2 Git Commit** | 60ceb870aea0777a6b18870f2bea5e3c55db6d49 |
| **Audit Date** | April 10th, 2022 |

# Claimed Smart Contract Features

| Claimed Feature Detail | Our Observation |
|---|---|
| **File 1: Remnant Token**<br>• Name: Remnant<br>• Symbol: REMN<br>• Decimals: 18<br>• Total supply: 10 billion<br>• Initial Amount Team: 10%<br>• Initial Amount Staking: 15%<br>• Initial Amount Ecosystem: 25%<br>• Initial Amount Other: 50%<br>• The owner can enable/disable bot protection, which is limiting the trades while adding liquidity.<br>• Bot protection can be applied to one trading pair at a time. Bot protection can not be applied to multiple trading pairs at a time. | **YES, This is valid.**<br><br>**Owner authorized wallet can set some percentage value and we suggest handling the private key of that wallet securely.** |
| **File 2: Staking Vested**<br>• Staking rewards<br>  ○ 7 days lock: 0% reward<br>  ○ 30 days lock: 10% reward<br>  ○ 90 days lock: 40% reward<br>  ○ 180 days lock: 100% reward<br>• Owner can update rewards per block | **YES, This is valid.**<br><br>**Owner authorized wallet can set some percentage value and we suggest handling the private key of that wallet securely.** |

# Audit Summary

According to the standard audit assessment, Customer`s solidity smart contracts are **"Secured"**. These contracts do contain owner control, which does not make them fully decentralized.


We used various tools like Slither, Solhint and Remix IDE. At the same time this finding is based on critical analysis of the manual audit.
All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Audit overview section. General overview is presented in AS-IS section and all identified issues can be found in the Audit overview section.

**We found 0 critical, 0 high, 0 medium and 0 low and some very low level issues.**

**Investors Advice:** Technical audit of the smart contract does not guarantee the ethical nature of the project. Any owner controlled functions should be executed by the owner with responsibility. All investors/users are advised to do their due diligence before investing in the project.

# Technical Quick Stats

| Main Category | Subcategory | Result |
|---|---|---|
| Contract Programming | Solidity version not specified | Passed |
| | Solidity version too old | Passed |
| | Integer overflow/underflow | Passed |
| | Function input parameters lack of check | Passed |
| | Function input parameters check bypass | Passed |
| | Function access control lacks management | Passed |
| | Critical operation lacks event log | Passed |
| | Human/contract checks bypass | Passed |
| | Random number generation/use vulnerability | N/A |
| | Fallback function misuse | Passed |
| | Race condition | Passed |
| | Logical vulnerability | Passed |
| | Features claimed | Passed |
| | Other programming issues | Passed |
| Code Specification | Function visibility not explicitly declared | Passed |
| | Var. storage location not explicitly declared | Passed |
| | Use keywords/functions to be deprecated | Passed |
| | Unused code | Passed |
| Gas Optimization | “Out of Gas” Issue | Passed |
| | High consumption ‘for/while’ loop | Passed |
| | High consumption ‘storage’ storage | Passed |
| | Assert() misuse | Passed |
| Business Risk | The maximum limit for mintage not set | Passed |
| | “Short Address” Attack | Passed |
| | “Double Spend” Attack | Passed |

**Overall Audit Result:** **PASSED**

## Code Quality

This audit scope has 2 smart contract files. Smart contract contains Libraries, Smart contracts, inherits and Interfaces. This is a compact and well written smart contract.

The libraries in Remnant Protocol are part of its logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned a specific address and its properties / methods can be reused many times by other contracts in the Remnant Protocol.

The Remnant Protocol team has **not** provided scenario and unit test scripts, which would have helped to determine the integrity of the code in an automated way.

Code parts are commented on in the smart contracts. But Ethereum's NatSpec commenting style is recommended.

## Documentation

We were given a Remnant token and staking smart contract code in the form of a github weblink. The commit of that code is mentioned above in the table.

As mentioned above, code parts are well commented. And the contract is straightforward so it's easy to understand its programming logic.

## Use of Dependencies

As per our observation, the libraries are used in this smart contract infrastructure that are based on well known industry standard open source projects.

Apart from libraries, its functions are used in external smart contract calls.

# AS-IS overview

## RTestToken2.sol

### Functions

| Sl. | Functions | Type | Observation | Conclusion |
|---|---|---|---|---|
| 1 | constructor | write | Passed | No Issue |
| 2 | bpAddNewSwapPairPool | external | access only Owner | No Issue |
| 3 | bpToggleOnOff | external | access only Owner | No Issue |
| 4 | bpSetMaxGwei | external | access only Owner | No Issue |
| 5 | bpSetMaxBuyValue | external | access only Owner | No Issue |
| 6 | bpSetMaxSellValue | external | access only Owner | No Issue |
| 7 | bpAddSwapPairPool | external | access only Owner | No Issue |
| 8 | bpRemoveSwapPairPool | external | access only Owner | No Issue |
| 9 | bpDisablePermanently | external | access only Owner | No Issue |
| 10 | bpToggleTrading | external | access only Owner | No Issue |
| 11 | bpResetAllAddressesTimesTransacted | external | access only Owner | No Issue |
| 12 | testGasConsumation32 | external | access only Owner | No Issue |
| 13 | testGasConsumation256 | external | access only Owner | No Issue |
| 14 | _beforeTokenTransfer | internal | Passed | No Issue |
| 15 | name | read | Passed | No Issue |
| 16 | symbol | read | Passed | No Issue |
| 17 | decimals | read | Passed | No Issue |
| 18 | totalSupply | read | Passed | No Issue |
| 19 | balanceOf | read | Passed | No Issue |
| 20 | transfer | write | Passed | No Issue |
| 21 | allowance | read | Passed | No Issue |
| 22 | approve | write | Passed | No Issue |
| 23 | transferFrom | write | Passed | No Issue |
| 24 | increaseAllowance | write | Passed | No Issue |
| 25 | decreaseAllowance | write | Passed | No Issue |
| 26 | _transfer | internal | Passed | No Issue |
| 27 | _mint | internal | Passed | No Issue |
| 28 | _burn | internal | Passed | No Issue |
| 29 | _approve | internal | Passed | No Issue |

| | | | | |
|---|---|---|---|---|
| **30** | _beforeTokenTransfer | internal | Passed | No Issue |
| **31** | _afterTokenTransfer | internal | Passed | No Issue |
| **32** | owner | read | Passed | No Issue |
| **33** | onlyOwner | modifier | Passed | No Issue |
| **34** | renounceOwnership | write | access only Owner | No Issue |
| **35** | transferOwnership | write | access only Owner | No Issue |
| **36** | _setOwner | internal | Passed | No Issue |

## StakingVested.sol

### Functions

| Sl. | Functions | Type | Observation | Conclusion |
|---|---|---|---|---|
| **1** | constructor | write | Passed | No Issue |
| **2** | owner | read | Passed | No Issue |
| **3** | onlyOwner | modifier | Passed | No Issue |
| **4** | renounceOwnership | write | access only Owner | No Issue |
| **5** | transferOwnership | write | access only Owner | No Issue |
| **6** | _transferOwnership | internal | Passed | No Issue |
| **7** | balanceOf | external | Passed | No Issue |
| **8** | weightOf | external | Passed | No Issue |
| **9** | getDeposit | external | Passed | No Issue |
| **10** | getDepositsLength | external | Passed | No Issue |
| **11** | getPendingRewardOf | external | Passed | No Issue |
| **12** | getUnlockSpecs | read | Passed | No Issue |
| **13** | now256 | read | Passed | No Issue |
| **14** | blockNumber | read | Passed | No Issue |
| **15** | updateRewardPerBlock | external | access only Owner | No Issue |
| **16** | recoverERC20 | external | access only Owner | No Issue |
| **17** | sync | external | Passed | No Issue |
| **18** | stake | external | Passed | No Issue |
| **19** | unstake | external | Passed | No Issue |
| **20** | claimRewards | external | Passed | No Issue |
| **21** | emergencyWithdraw | external | Passed | No Issue |
| **22** | _sync | internal | Passed | No Issue |
| **23** | _stake | internal | Passed | No Issue |
| **24** | _unstake | internal | Passed | No Issue |
| **25** | _claimRewards | internal | Passed | No Issue |
| **26** | _transferTokenFrom | internal | Passed | No Issue |
| **27** | _safeTokenTransfer | internal | Passed | No Issue |

# Severity Definitions

| Risk Level | Description |
|---|---|
| Critical | Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc. |
| High | High-level vulnerabilities are difficult to exploit; however, they also have significant impact on smart contract execution, e.g. public access to crucial |
| Medium | Medium-level vulnerabilities are important to fix; however, they can't lead to tokens lose |
| Low | Low-level vulnerabilities are mostly related to outdated, unused etc. code snippets, that can't have significant impact on execution |
| Lowest / Code Style / Best Practice | Lowest-level vulnerabilities, code style violations and info statements can't affect smart contract execution and can be ignored. |


**Email: audit@EtherAuthority.io**

# Audit Findings

## Critical Severity

No Critical severity vulnerabilities were found.

## High Severity

No High severity vulnerabilities were found.

## Medium

No Medium severity vulnerabilities were found.

## Low

No Low severity vulnerabilities were found.

## Very Low / Informational / Best practices:

(1) Unstaking does not reduce *Deposit Length* in StakingVested.sol

```
delete user.deposits[_depositId];
```

The function _unstake deletes the user's staking entry from the *deposits* array. This actually does not reduce the getDepositLength value. So in other words, the getDepositLength function will always return total stakes made by the user even if some are unstaked. It will not return the active stakes.

More specifically, deleting an array index creates an empty array index leaving the total length of the array as it is.

Resolution: If this is desired behavior, then this point can be safely ignored, as this does not create any security or logical vulnerability. On another hand, if getDepositLength function is expected to return all active stake of the user, then this empty array index should be replaced by the last array element.

(2) Inconsistency can be improved

```
// update user record
user.tokenAmount -= _amount;
user.totalWeight = user.totalWeight - _weight;
user.totalRewardsClaimed += _rewardAmount;
```

This is not any major issue and can be safely ignored. However, it is good practice to maintain the consistency in the code writing style. So,

*user.totalWeight = user.totalWeight - _weight;*

can be replaced with,

*user.totalWeight -= _weight;*

So, it can be consistent with other code lines.

# Centralization

This smart contract has some functions which can be executed by the Admin (Owner) only. If the admin wallet private key would be compromised, then it would create trouble. Following are Admin functions:

- bpAddNewSwapPairPool: RemnantToken owner can add a new swap pair pool whitelist address to the bot protection.
- bpToggleOnOff: RemnantToken owner can toggles bot protection, blocking suspicious transactions during liquidity events.
- bpSetMaxGwei: RemnantToken owner can set max gwei allowed in transaction when bot protection is on.
- bpSetMaxBuyValue: RemnantToken owner can set max buy value when bot protection is on.
- bpSetMaxSellValue: RemnantToken owner can set max sell value when bot protection is on.
- bpAddSwapPairPool: RemnantToken owner can add swap pair pool address.
- bpRemoveSwapPairPool: RemnantToken owner can remove swap pair pool address.
- bpDisablePermanently: RemnantToken owner can turn off bot protection permanently.
- bpToggleTrading: RemnantToken owner can toggle trading.
- bpResetAllAddressesTimesTransacted: RemnantToken owner can reset transaction count of all addresses.
- testGasConsumation32: RemnantToken owner can set gas consumption value.
- testGasConsumation256: RemnantToken owner can set gas consumption value.
- updateRewardPerBlock: StakingVested owner can update reward block values.
- recoverERC20: StakingVested owner can recover lost tokens that find their way to the staking contract.

# Conclusion

We were given a contract code. And we have used all possible tests based on given objects as files. We have not observed any major issues. So, **it's good to go for the mainnet deployment**.

Since possible test cases can be unlimited for such smart contracts protocol, we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover maximum possible test cases to scan everything.

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. Smart Contract's high-level description of functionality was presented in the As-is overview section of the report.

Audit report contains all found security vulnerabilities and other issues in the reviewed code.

Security state of the reviewed contract, based on standard audit procedure scope, is **"Secured"**.

# Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

**Manual Code Review:**

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

**Vulnerability Analysis:**

Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

**Documenting Results:**

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyze the feasibility of an attack in a live system.

**Suggested Solutions:**

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

# Disclaimers

## EtherAuthority.io Disclaimer

EtherAuthority team has analyzed this smart contract in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only. We also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

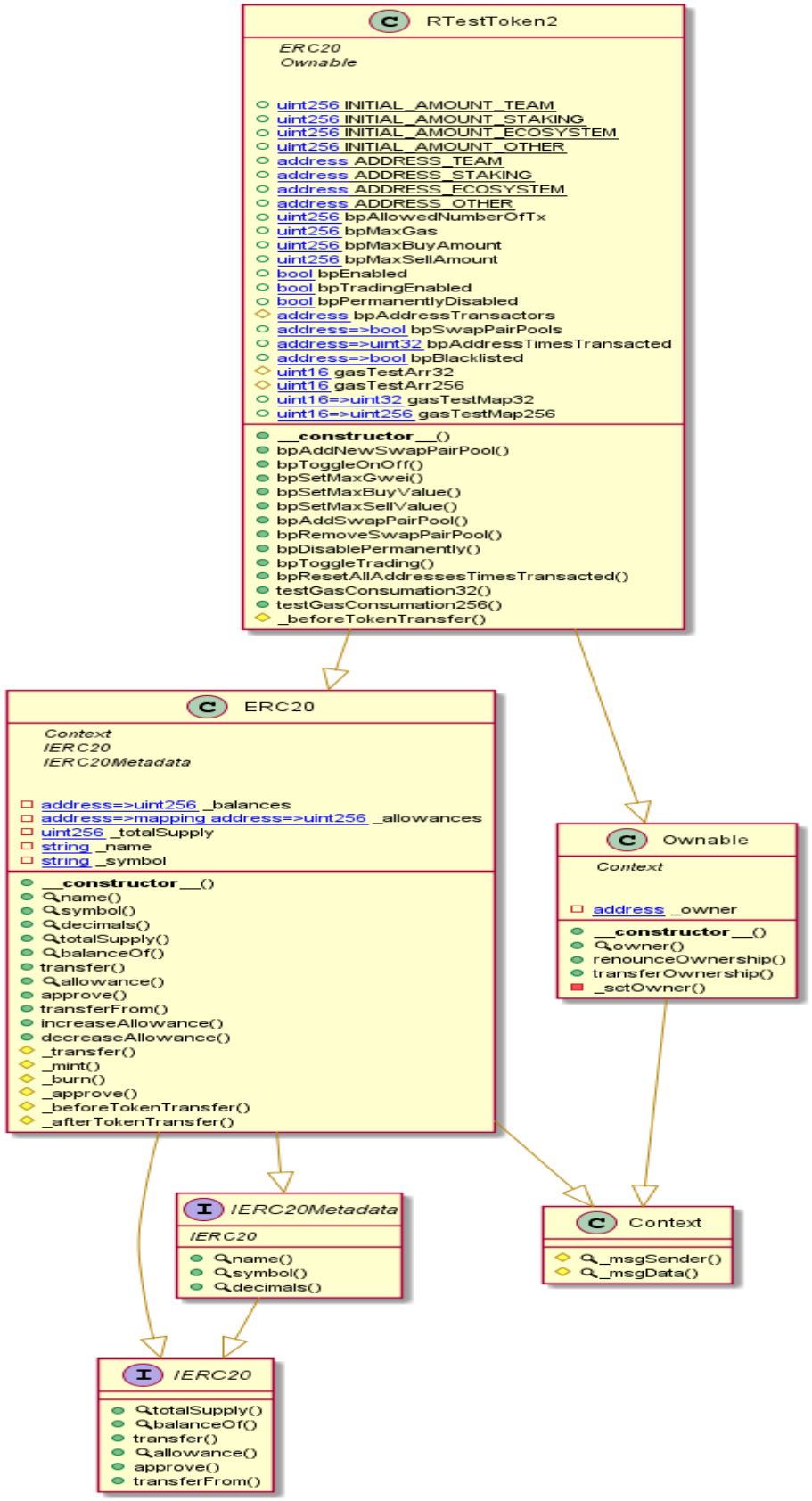
## Technical Disclaimer

Smart contracts are deployed and executed on the blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can’t guarantee explicit security of the audited smart contracts.

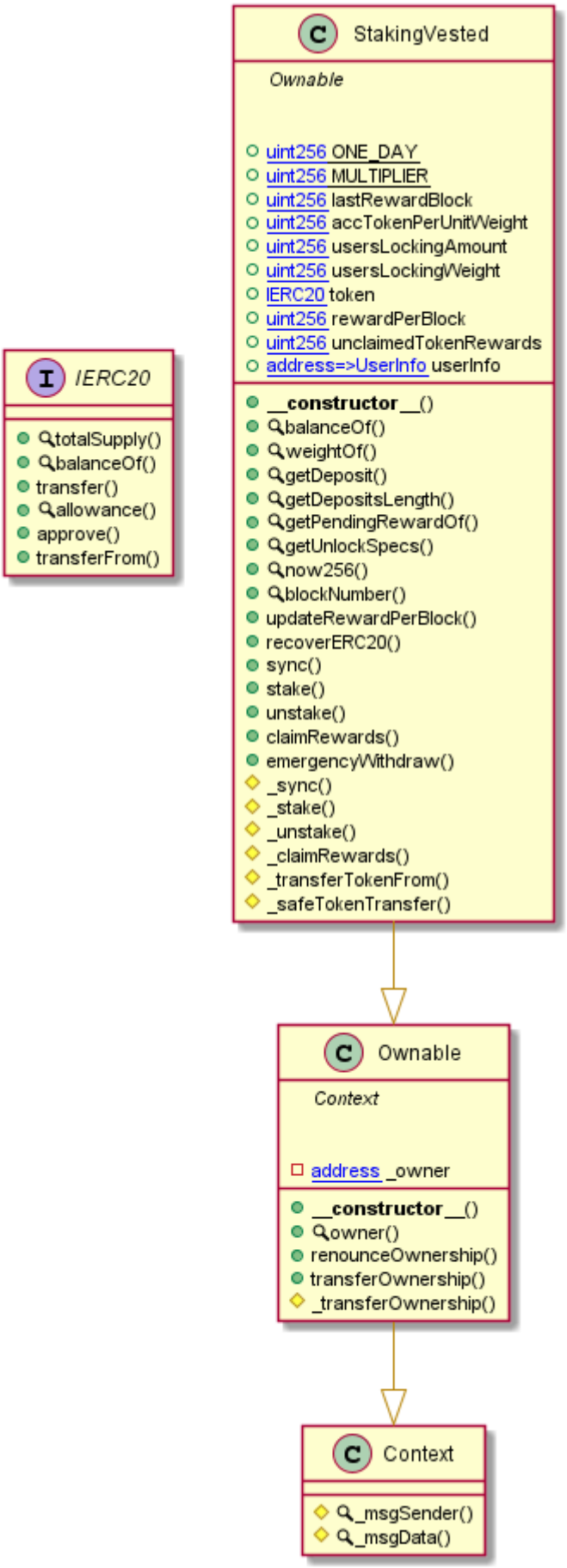
# Appendix

## Code Flow Diagram - Remnant Token

## RTestToken2 Diagram



# StakingVestedDiagram






**Email: audit@EtherAuthority.io**

# Slither Results Log

## Slither log >> RTestToken2.sol

```
INFO:Detectors:
RTestToken2.bpAddressTransactors (RTestToken2.sol#492) is never initialized. It is used in:
        - RTestToken2.bpResetAllAddressesTimesTransacted() (RTestToken2.sol#584-588)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-state-variables
```

```
INFO:Detectors:
RTestToken2._beforeTokenTransfer(address,address,uint256) (RTestToken2.sol#611-640) compares to a boolean constant:
        -bpSwapPairPools[from] == true (RTestToken2.sol#618)
RTestToken2._beforeTokenTransfer(address,address,uint256) (RTestToken2.sol#611-640) compares to a boolean constant:
        -bpSwapPairPools[to] == true (RTestToken2.sol#628)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality
INFO:Detectors:
Context._msgData() (RTestToken2.sol#92-94) is never used and should be removed
ERC20._burn(address,uint256) (RTestToken2.sol#336-351) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version0.8.4 (RTestToken2.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.
solc-0.8.4 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
name() should be declared external:
        - ERC20.name() (RTestToken2.sol#123-125)
symbol() should be declared external:
        - ERC20.symbol() (RTestToken2.sol#131-133)
decimals() should be declared external:
        - ERC20.decimals() (RTestToken2.sol#148-150)
totalSupply() should be declared external:
        - ERC20.totalSupply() (RTestToken2.sol#155-157)
balanceOf(address) should be declared external:
        - ERC20.balanceOf(address) (RTestToken2.sol#162-164)
transfer(address,uint256) should be declared external:
        - ERC20.transfer(address,uint256) (RTestToken2.sol#174-177)
allowance(address,address) should be declared external:
        - ERC20.allowance(address,address) (RTestToken2.sol#182-184)
approve(address,uint256) should be declared external:
        - ERC20.approve(address,uint256) (RTestToken2.sol#193-196)
transferFrom(address,address,uint256) should be declared external:
        - ERC20.transferFrom(address,address,uint256) (RTestToken2.sol#211-225)
increaseAllowance(address,uint256) should be declared external:
        - ERC20.increaseAllowance(address,uint256) (RTestToken2.sol#239-242)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20.decreaseAllowance(address,uint256) (RTestToken2.sol#258-266)
```

```
decreaseAllowance(address,uint256) should be declared external:
        - ERC20.decreaseAllowance(address,uint256) (RTestToken2.sol#258-266)
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (RTestToken2.sol#452-454)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (RTestToken2.sol#460-463)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:RTestToken2.sol analyzed (6 contracts with 75 detectors), 20 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

## Slither log >> RTestToken2.sol

```
INFO:Detectors:
Reentrancy in StakingVested._claimRewards(address,uint256) (StakingVested.sol#389-414):
        External calls:
        - _safeTokenTransfer(_staker,_rewardAmount) (StakingVested.sol#412)
                - IERC20(token).transfer(_to,tokenBal) (StakingVested.sol#424)
                - IERC20(token).transfer(_to,_amount) (StakingVested.sol#426)
        Event emitted after the call(s):
        - Claimed(_staker,_rewardAmount) (StakingVested.sol#413)
Reentrancy in StakingVested._stake(address,uint256,uint256) (StakingVested.sol#317-346):
        External calls:
        - _transferTokenFrom(address(_staker),address(this),_amount) (StakingVested.sol#323)
                - IERC20(token).transferFrom(_from,_to,_value) (StakingVested.sol#417)
        Event emitted after the call(s):
        - Staked(_staker,_amount) (StakingVested.sol#345)
Reentrancy in StakingVested._unstake(address,uint256,bool) (StakingVested.sol#348-387):
        External calls:
        - _safeTokenTransfer(_staker,tokenToSend) (StakingVested.sol#385)
                - IERC20(token).transfer(_to,tokenBal) (StakingVested.sol#424)
                - IERC20(token).transfer(_to,_amount) (StakingVested.sol#426)
        Event emitted after the call(s):
        - Unstaked(_staker,_amount) (StakingVested.sol#386)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
StakingVested._sync() (StakingVested.sol#299-315) uses timestamp for comparisons
        Dangerous comparisons:
        - _weightLocked == 0 (StakingVested.sol#304)
        - require(bool,string)(surplusToken >= tokenReward,TKNStaking: Insufficient TKN tokens for rewards) (StakingVested.s
#311)
StakingVested._unstake(address,uint256,bool) (StakingVested.sol#348-387) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(now256() > stakeDeposit.lockedUntil,TKNStaking: Deposit not unlocked yet) (StakingVested.sol#
7)
StakingVested._safeTokenTransfer(address,uint256) (StakingVested.sol#421-428) uses timestamp for comparisons
        Dangerous comparisons:
        - _amount > tokenBal (StakingVested.sol#423)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
```

```
INFO:Detectors:
Context._msgData() (StakingVested.sol#84-86) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version0.8.4 (StakingVested.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.
6
solc-0.8.4 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Parameter StakingVested.balanceOf(address)._user (StakingVested.sol#196) is not in mixedCase
Parameter StakingVested.weightOf(address)._user (StakingVested.sol#201) is not in mixedCase
Parameter StakingVested.getDeposit(address,uint256)._user (StakingVested.sol#206) is not in mixedCase
Parameter StakingVested.getDeposit(address,uint256)._depositId (StakingVested.sol#206) is not in mixedCase
Parameter StakingVested.getDepositsLength(address)._user (StakingVested.sol#211) is not in mixedCase
Parameter StakingVested.getPendingRewardOf(address,uint256)._staker (StakingVested.sol#215) is not in mixedCase
Parameter StakingVested.getPendingRewardOf(address,uint256)._depositId (StakingVested.sol#215) is not in mixedCase
Parameter StakingVested.getUnlockSpecs(uint256,uint256)._amount (StakingVested.sol#233) is not in mixedCase
Parameter StakingVested.getUnlockSpecs(uint256,uint256)._lockMode (StakingVested.sol#233) is not in mixedCase
Parameter StakingVested.updateRewardPerBlock(uint256)._newRewardPerBlock (StakingVested.sol#263) is not in mixedCase
Parameter StakingVested.recoverERC20(address,uint256)._tokenAddress (StakingVested.sol#269) is not in mixedCase
Parameter StakingVested.recoverERC20(address,uint256)._tokenAmount (StakingVested.sol#269) is not in mixedCase
Parameter StakingVested.stake(uint256,uint256)._amount (StakingVested.sol#280) is not in mixedCase
Parameter StakingVested.stake(uint256,uint256)._lockMode (StakingVested.sol#280) is not in mixedCase
Parameter StakingVested.unstake(uint256)._depositId (StakingVested.sol#285) is not in mixedCase
Parameter StakingVested.claimRewards(uint256)._depositId (StakingVested.sol#290) is not in mixedCase
Parameter StakingVested.emergencyWithdraw(uint256)._depositId (StakingVested.sol#295) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (StakingVested.sol#123-125)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (StakingVested.sol#131-134)
blockNumber() should be declared external:
        - StakingVested.blockNumber() (StakingVested.sol#258-261)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:StakingVested.sol analyzed (4 contracts with 75 detectors), 35 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

# Solidity Static Analysis

**RTestToken2.sol**

## Gas & Economy

### Gas costs:

Gas requirement of function ERC20.name is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 135:4:

### Gas costs:

Gas requirement of function RTestToken2.bpResetAllAddressesTimesTransacted is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 596:4:

### Gas costs:

Gas requirement of function RTestToken2.testGasConsumation256 is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 611:4:

### For loop over dynamic array:

Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.
more
Pos: 597:8:

## Miscellaneous

### Constant/View/Pure functions:

IERC20.transfer(address,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 25:4:

### Constant/View/Pure functions:

ERC20._afterTokenTransfer(address,address,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 424:4:

### Similar variable names:

ERC20._burn(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.

Pos: 362:49:

### Guard conditions:

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

more

Pos: 589:8:

### Guard conditions:

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

more

Pos: 626:12:

### No return:

IERC20Metadata.symbol(): Defines a return type but never explicitly returns a value.
Pos: 91:4:

### No return:

IERC20Metadata.decimals(): Defines a return type but never explicitly returns a value.
Pos: 96:4:

### Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 642:16:

### Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 631:16:

### Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 632:16:

**StakingVested.sol**

## Security

### Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in StakingVested._sync(): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 299:4:

### Block timestamp:

Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.

more

Pos: 255:15:

## Gas & Economy

### Gas costs:

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

Pos: 215:4:

### Gas costs:

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

Pos: 295:4:

## Miscellaneous

### Constant/View/Pure functions:

StakingVested.getUnlockSpecs(uint256,uint256) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 233:4:

### No return:

IERC20.transferFrom(address,address,uint256): Defines a return type but never explicitly returns a value.
Pos: 72:4:

### Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 357:8:

### Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 397:8:

### Data truncated:

Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.
Pos: 400:33:

# Solhint Linter

**RTestToken2.sol**

```
RTestToken2.sol:5:6: Error: Parse error: missing 'constant' at
'SwapNotEnabledYet'
RTestToken2.sol:5:23: Error: Parse error: missing '=' at '('
RTestToken2.sol:232:18: Error: Parse error: missing ';' at '{'
RTestToken2.sol:273:18: Error: Parse error: missing ';' at '{'
RTestToken2.sol:306:18: Error: Parse error: missing ';' at '{'
RTestToken2.sol:355:18: Error: Parse error: missing ';' at '{'
RTestToken2.sol:635:44: Error: Parse error: mismatched input '('
expecting {';', '='}
RTestToken2.sol:645:44: Error: Parse error: mismatched input '('
expecting {';', '='}
```

**StakingVested.sol**

```
StakingVested.sol:2:1: Error: Compiler version 0.8.4 does not satisfy
the r semver requirement
StakingVested.sol:97:5: Error: Explicitly mark visibility in function
(Set ignoreConstructors to true if using solidity >=0.7.0)
StakingVested.sol:188:5: Error: Explicitly mark visibility in
function (Set ignoreConstructors to true if using solidity >=0.7.0)
StakingVested.sol:255:16: Error: Avoid to make time-based decisions
in your business logic
```

**Software analysis result:**

These software reported many false positive results and some are informational issues. So, those issues can be safely ignored.




**Email: audit@EtherAuthority.io**