

# SMART CONTRACT

---

## Security Audit Report

Customer:	OneSeed
Website:	<a href="https://oneseed.app">https://oneseed.app</a>
Platform:	Ethereum
Language:	Solidity
Date:	July 6th, 2021

# Table of contents

Introduction .....	4
Project Background .....	4
Audit Scope .....	4
Claimed Smart Contract Features .....	5
Audit Summary .....	6
Technical Quick Stats .....	7
Code Quality .....	8
Documentation .....	8
Use of Dependencies .....	8
AS-IS overview .....	9
Severity Definitions .....	13
Audit Findings .....	12
Conclusion .....	23
Our Methodology .....	24
Disclaimers .....	26
Appendix	
• Code Flow Diagram .....	27
• Slither Report Log .....	28

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 PUBLIC AFTER ISSUES ARE RESOLVED.

# Introduction

EtherAuthority was contracted by the OneSeed team to perform the Security audit of the OneSeed Token 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 July 6th, 2021.

**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

The goal of the OneSeed contract is to develop an ERC20-based token. There is a stack component and a referral component to allow users to stake their tokens and receive a return based on the amount in staking. The returns are used to remunerate the team, partners and liquidity.

## Audit scope

<b>Name</b>	<b>Code Review and Security Analysis Report for OneSeed token Smart Contract</b>
<b>Platform</b>	<b>Ethereum / Solidity</b>
<b>File</b>	OneSeed.sol
<b>Smart Contract Online Code</b>	<a href="https://rinkeby.etherscan.io/address/0x53ACF45301969619117C47981A4fc0eb71bc9aB6#code">https://rinkeby.etherscan.io/address/0x53ACF45301969619117C47981A4fc0eb71bc9aB6#code</a>
<b>File MD5 Hash</b>	274549CDA44784DD3F27E209B1BBBBB4F
<b>Audit Date</b>	July 6th, 2021
<b>Updated File MD5 Hash</b>	8DBEBD8D5572DD5E34C8F31A35DB9E28
<b>Final Audit Date</b>	August 14th, 2021

## Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Name: OneSeed	<b>YES, This is valid.</b>
Symbol: SEED	<b>YES, This is valid.</b>
Decimal: 2	<b>YES, This is valid.</b>
Liquidity Fee: 5%	<b>YES, This is valid.</b>
<ul style="list-style-type: none"><li>• The Owner can set Tax fee percentage, Liquidity Fee percentage, max tax percentage, Swap And Liquify Enable status, enableTrading, etc.</li><li>• lockStake: The ActiveStaker can check if the stake is already locked or not.</li><li>• extendStakeLock: The ActiveStaker can extend stack locked time.</li></ul>	<b>YES, This is valid. The smart contract owner controls these functions, so the owner must handle the private key of the owner's wallet very securely. Because if the private key is compromised, then it will create problems.</b>

## Audit Summary

According to the standard audit assessment, the Customer's solidity smart contract is **"Secured"**. These contracts also have owner functions (described in the centralization section below), which does not make everything 100% decentralized. Thus, the owner must execute those smart contract functions as per the business plan.



**We found 2 critical, 0 high, 5 medium and 8 low and some very low level technical issues.**

**They have fixed/acknowledged in the revised contract code.**

**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	Moderated
	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	Passed
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	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
	Other code specification issues	Passed
Gas Optimization	"Out of Gas" Issue	Passed
	High consumption 'for/while' loop	Moderated
	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 1 smart contract. This smart contract also contains Libraries, Smart contracts inherits and Interfaces. This is a compact and well written contract.

The libraries in the OneSeed Token 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 OneSeed Token .

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

Some code parts are **not well** commented on smart contracts.

## Documentation

We were given a OneSeed Token smart contract code in the form of an Etherscan web link. The hashes of that code are mentioned above in the table.

As mentioned above, some code parts are **not well** commented. So it is difficult to quickly understand the programming flow as well as complex code logic. Comments are very helpful in understanding the overall architecture of the protocol.

## 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. And their core code blocks are written well.

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



# AS-IS overview

OneSeed Token is a smart contract, having functionality like swap And Liquify, stake, etc.

## OneSeed.sol

### (1) Interface

- (a) IERC20
- (b) IUniswapV2Factory
- (c) IUniswapV2Pair
- (d) IUniswapV2Router01
- (e) IUniswapV2Router02

### (2) Inherited contracts

- (a) Context
- (b) Ownable

### (3) Struct

- (a) Stake
- (b) StakeReward
- (c) PartnerReward

### (4) Usages

- (a) using SafeMath for uint256;
- (b) using Address for address;

### (5) Events

- (a) event MinTokensBeforeSwapUpdated(uint256 minTokensBeforeSwap);
- (b) event SwapAndLiquifyEnabledUpdated(bool enabled);
- (c) event SwapAndLiquify(uint256 tokensSwapped,uint256 ethReceived,uint256 tokensIntoLiquidity);

## (6) Functions

Sl.	Functions	Type	Observation	Conclusion
1	name	read	Passed	No Issue
2	symbol	read	Passed	No Issue
3	decimals	read	Passed	No Issue
4	totalSupply	read	Passed	No Issue
5	balanceOf	read	Passed	No Issue
6	transfer	write	Passed	No Issue
7	allowance	read	Passed	No Issue
8	approve	write	Passed	No Issue
9	transferFrom	write	Passed	No Issue
10	increaseAllowance	write	Passed	No Issue
11	decreaseAllowance	write	Passed	No Issue
12	isExcludedFromReward	read	External Visibility should be preferred	Refer Audit Findings
13	totalFees	read	External Visibility should be preferred	Refer Audit Findings
14	deliver	write	External Visibility should be preferred	Refer Audit Findings
15	reflectionFromToken	read	External Visibility should be preferred	Refer Audit Findings
16	tokenFromReflection	read	External Visibility should be preferred	Refer Audit Findings
17	excludeFromReward	write	External Visibility should be preferred	Refer Audit Findings
18	includeInReward	external	Passed	No Issue
19	_transferBothExcluded	write	Passed	No Issue
20	excludeFromFee	write	External Visibility should be preferred	Refer Audit Findings
21	includeInFee	write	External Visibility should be preferred	Refer Audit Findings
22	setTaxFeePercent	external	Absence of Input Validation	Refer Audit Findings
23	setLiquidityFeePercent	external	Absence of Input Validation	Refer Audit Findings
24	setMaxTxPercent	external	access only Owner	No Issue
25	setSwapAndLiquifyEnabled	write	External Visibility should be preferred	Refer Audit Findings
26	enableTrading	external	access only Owner	No Issue
27	_getValues	read	Passed	No Issue
28	_getTValues	read	Passed	No Issue
29	_getRValues	write	Passed	No Issue
30	_getRate	read	Passed	No Issue
31	_getCurrentSupply	read	Loops are extremely costly	Refer Audit Findings
32	_takeLiquidity	write	Passed	No Issue
33	calculateTaxFee	read	Passed	No Issue

34	calculateLiquidityFee	read	Passed	No Issue
35	removeAllFee	write	Passed	No Issue
36	restoreAllFee	write	Passed	No Issue
37	isExcludedFromFee	read	External Visibility should be preferred	Refer Audit Findings
38	_approve	write	Passed	No Issue
39	_transfer	write	Passed	No Issue
40	swapAndLiquify	write	Passed	No Issue
41	swapTokensForEth	write	Passed	No Issue
42	addLiquidity	write	Passed	No Issue
43	_tokenTransfer	write	Passed	No Issue
44	_transferStandard	write	Passed	No Issue
45	_transferToExcluded	write	Passed	No Issue
46	_transferFromExcluded	write	Passed	No Issue
47	stakeReferred	write	Passed	No Issue
48	stake	write	Passed	No Issue
49	lockStake	write	Passed	No Issue
50	extendStakeLock	write	Passed	No Issue
51	isStaking	read	User will remain marked as STAKER even after Unstaking	Refer Audit Findings
52	unstake	write	access Only ActiveStaker	No Issue
53	getStakingMember	write	User will remain marked as STAKER even after Unstaking	Refer Audit Findings
54	isUnstakePermitted	read	Passed	No Issue
55	rewardAllStaking	write	Passed	No Issue
56	adjustInterest	write	Passed	No Issue
57	isReferred	read	Passed	No Issue
58	addToAddress	write	Passed	No Issue
59	removeFromAddress	write	Passed	No Issue
60	setTeamAddress	write	Passed	No Issue
61	getMyStakeReward	read	Passed	No Issue
62	getMyPartnerReward	read	Passed	No Issue
63	isAutoCompound	read	Passed	No Issue
64	toggleAutoCompound	write	Passed	No Issue
65	getCountMyPartnerRewards	read	Functions could be marked internal	Refer Audit Findings
66	getCountMyStakeRewards	read	Functions could be marked internal	Refer Audit Findings
67	getGlobalStakingAmount	read	Passed	No Issue
68	setMinimumStakingAmount	write	Absence of Error messages in Require	Refer Audit Findings
69	setRewardsPerCall	write	Passed	No Issue
70	setGlobalMaxStakingAmount	write	Passed	No Issue
71	OnlyActiveStaker	modifier	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 functions
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.

## Audit Findings

### Critical

(1) Team Reward is never added during Reward distribution: Incorrect Solidity Code

```
} else {  
    // if no referer than add the amount to team  
    tr.add(_partnerReward);  
}
```

As per the current design of the **rewardAllStaking** function, for those stakers with referral address, their **partner reward** is being added to the team.

In order to achieve this behavior from the **rewardAllStaking function**, the following logic has been written in the function body at Line no - **1388 - 1391**

This is an incorrect logic because it simply performs an additional operation(using Safemath library's add function) where it adds the **\_partnerReward** to the local variable **tr**.

However, the result of this addition is never really stored in the **local variable tr**.

This will lead to a completely unexpected scenario where the team reward will only be **2.5%** instead of **7.5%**(Team Reward + Partner Reward) as the partner reward percentage was never added and stored adequately.

**Resolution:** The above-mentioned ELSE block at line number - 1388 - 1391 should be modified as follows:

```
    } else {  
        // if no referer than add the amount to team  
        tr = tr.add(_partnerReward);  
    }
```

**Status: Fixed.**

(2) User will remain marked as STAKER even after Unstaking:

```
function isStaking(address a) public view returns (bool){  
    return _stakeholders[a].isStaker;  
}
```

The **Stake** struct in the protocol has boolean data, i.e., **isStaker**. This boolean value is assigned **true** whenever a user Stakes in his tokens in the contract.

However, the **isStaker** value is never marked as **FALSE**, even after the user has **unstaked** all of his tokens and is no longer a staker.

This might lead to an unexpected scenario where wrong information is sent out to users when the execute functions like **getStakingMember** or **isStaking**.

**Resolution:** Accurate data should be stored on-chain. Unstake should be modified accordingly.

**Note:** If this is part of the plan then please ignore this.

**Status: Acknowledged.**

**High**

No high severity vulnerabilities were found.

## Medium

(1) Loops are extremely costly:

```
function includeInReward(address account) external onlyOwner() {
    require(!_isExcluded[account], "Account is already excluded");
    for (uint256 i = 0; i < _excluded.length; i++) {
        if (_excluded[i] == account) {
            _excluded[i] = _excluded[_excluded.length - 1];
            _tOwned[account] = 0;
            _isExcluded[account] = false;
            _excluded.pop();
            break;
        }
    }
}
```

```
function _getCurrentSupply() private view returns (uint256, uint256) {
    uint256 rSupply = _rTotal;
    uint256 tSupply = _tTotal;
    for (uint256 i = 0; i < _excluded.length; i++) {
        if (_rOwned[_excluded[i]] > rSupply || _tOwned[_excluded[i]] > tSupply) return (_rTotal, _tTotal);
        rSupply = rSupply.sub(_rOwned[_excluded[i]]);
        tSupply = tSupply.sub(_tOwned[_excluded[i]]);
    }
    if (rSupply < _rTotal.div(_tTotal)) return (_rTotal, _tTotal);
    return (rSupply, tSupply);
}
```

The **OneSeed** contract has some **for loops** in the contract that include state variables like `.length` of a non-memory array, in the condition of the for loops.

As a result, these state variables consume a lot more extra gas for every iteration of the for loop.

**Resolution:** It's quite effective to use a local variable instead of a state variable like `.length` in a loop. This will be a significant step in optimizing gas usage. Also, if owner excludes very limited wallets, then this issue can be safely ignored.

**Status:** **Acknowledged.**

(2) Inadequate input validation while setting minimum Staking Amount:

```
function setMinimumStakingAmount(uint256 min) public restrictedOwner {
    require(min < _globalMaxStakingAmount);
    _minimumStakingAmount = min;
}
```

The **setMinimumStakingAmount** function allows the owner to set the minimum staking amount.

However, the **required** statements only validate that the argument passed should be less than **\_globalMaxStakingAmount**. It doesn't really restrict the owner from passing a zero value which might result in an unexpected scenario. Thus failing to impose an adequate input validation mechanism.

**Resolution:** In order to avoid the above-mentioned scenario, a require statement should be included as follows:

**Status:** Fixed.

```
function setMinimumStakingAmount(uint256 min) public restrictedOwner {
    require(min < _globalMaxStakingAmount && min > 0,
        "ERROR MSG: Argument should be more than ZERO & less than Global Max");
    _minimumStakingAmount = min;
}
```

**Status:** Fixed.

(3) AdjustInterest function has been marked public without any access restriction modifier:

```
// adjust interest after 24 hours (less interest every day)
function adjustInterest() public {
    if (now.sub(_lastInterestAdjustment) >= _rewardTime) {
        _lastStakingInterest = _stakingInterest;
        _stakingInterest = _stakingInterest.sub((_stakingInterest.mul(_interestModification)
        _lastInterestAdjustment = now;
    }
}
```

The **adjustInterest** function includes quite an imperative task of adjusting interest rates as per the intended plan. However, this function has simply been marked as public and made accessible to all the users.

While the IF statement does take care of the time logic very well, it would still be considered a better practice to assign an access restriction modifier(like **restrictedOwner** modifier in the contract) to functions like these in order to avoid any unexpected behavior



in the future.

**Status:** Fixed.

(4) Multiplication is being performed on the result of Division:

```
function rewardAllStaking(uint256 maxPerCall) public {
    // function should not be running if no one has joined staking yet
    require(_stakingId > 0, 'No staking members yet');

    // maxPerCall indicates and how many stakers receive their rewards in this call.
    if (maxPerCall == 0 || maxPerCall > _rewardsPerCall) {
        maxPerCall = _rewardsPerCall;
    }

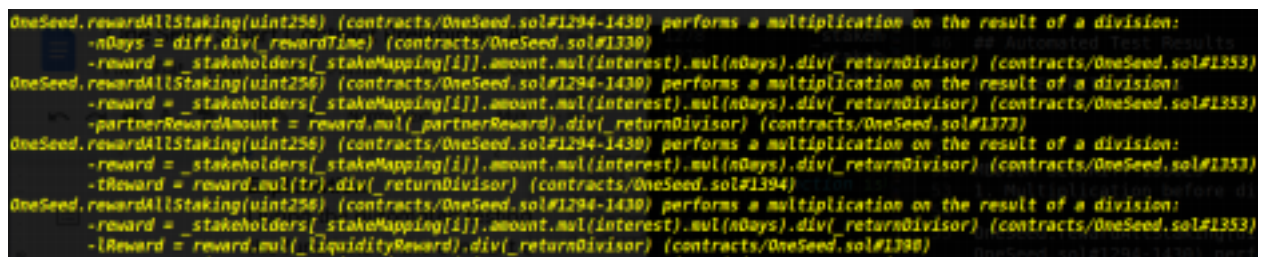
    // Lower the interest if 24 hours passed
    adjustInterest();
```

During the automated testing of the **OneSeed** contract, it was found that some of the functions in the contract are performing multiplication on the result of a Division. Integer Divisions in Solidity might be truncated. Moreover, this performing division before multiplication might lead to loss of precision.

The following functions involve division before multiplication in the mentioned lines:

- **rewardAllStaking()** at Line 1294-1430

#### Automated Test Result



The screenshot displays automated test results for the OneSeed contract. It shows several lines of code where the rewardAllStaking function is called, and each call is annotated with a comment stating: "OneSeed.rewardAllStaking(uint256) (contracts/OneSeed.sol#1294-1430) performs a multiplication on the result of a division:". The code snippets include calculations for nDays, reward, partnerRewardAmount, and tReward, all involving division before multiplication.

The following functions involve division before multiplication in the mentioned lines

**Resolution:** Solidity doesn't encourage arithmetic operations that involve division before multiplication. Therefore the above-mentioned functions should be checked once and redesigned if they do not lead to expected results.

**Status:** Fixed.



(5) Require statements should be preferred instead of IF Statement :

```
// adjust interest after 24 hours (less interest every day)
function adjustInterest() public {
    if (now.sub(_lastInterestAdjustment) >= _rewardTime) {
        _lastStakingInterest = _stakingInterest;
        _stakingInterest = _stakingInterest.sub((_stakingInterest.mul(_interestModification)
        _lastInterestAdjustment = now;
    }
}
```

As per the current design of the **adjustInterest** function, it is a strict requirement that the time must be more than 24 hours since the last interest adjustment, in order to execute this function successfully.

Therefore, it is considered a better practice to use **require statements** for such strict validations in a function.

While **requiring** statements increases code readability, it also enables us to understand the exact reason behind a particular transaction revert. .

**Resolution:** Require statement should be used instead of IF Statement, as follows.

```
// adjust interest after 24 hours (less interest every day)
function adjustInterest() public {
    require (now.sub(_lastInterestAdjustment) >= _rewardTime,"Error MSg: Interest can't be adjusted before 24 hours." ) ;

    _lastStakingInterest = _stakingInterest;
    _stakingInterest = _stakingInterest.sub((_stakingInterest.mul(_interestModification).div(_returnDivisor)));
    _lastInterestAdjustment = now;
}
```

**Status:** Acknowledged.

**Low**

(1) Redundant State Variable Update:

```
uint256 public _taxFee = 0;
bool public tradingEnabled = false;
```

The OneSeed Smart contract involves redundant updating of some of the state variables in the contract.

A boolean variable is by-default initialized to FALSE whereas a uint256 is initialized to ZERO.

Hence, such state variables do not need to be initialized explicitly.

**Resolution:** Redundant initialization of state variables should be avoided.

**Status:** **Acknowledged.**

(2) Absence of Input Validation :

```
function setTaxFeePercent(uint256 taxFee) external onlyOwner() {
    _taxFee = taxFee;
}

function setLiquidityFeePercent(uint256 liquidityFee) external onlyOwner() {
    _liquidityFee = liquidityFee;
}
```

The OneSeed Smart contract involves redundant updating of some of the state variables in the contract.

**Resolution:** The above-mentioned functions should include required statements to ensure only valid arguments are passed to these functions.

**Status:** **Acknowledged.**

(3) No Events emitted after imperative State Variable modification:

```
function setTaxFeePercent(uint256 taxFee) external onlyOwner() {
    _taxFee = taxFee;
}

function setLiquidityFeePercent(uint256 liquidityFee) external onlyOwner() {
    _liquidityFee = liquidityFee;
}
```

Functions that update an imperative arithmetic state variable contract should emit an event after the update.

The following functions modify some crucial arithmetic parameters like **\_taxFee**, **\_liquidityFee**:

- **setTaxFeePercent**
- **setLiquidityFeePercent**

**Resolution:** Redundant initialization of state variables should be avoided.

**Status:** **Acknowledged.**

(4) Absence of Error messages in Require:

```
function setMinimumStakingAmount(uint256 min) public restrictedOwner {  
    require(min < _globalMaxStakingAmount);  
    _minimumStakingAmount = min;  
}
```

The **OneSeed** contract includes a few functions(*at the above-mentioned lines*) that don't contain any error message in the **required** statement.

While this makes it troublesome to detect the reason behind a particular function revert, it also reduces the readability of the code.

**Resolution:** Error Messages must be included in every required statement in the contract.

**Status:** **Acknowledged.**

(5) No Zero Address validation in setTEamAddress function:

The **OneSeed** Contract includes a function that updates an imperative address in the contract like **\_teamAddress** etc.

However, during the automated testing of the contract it was found that no Zero Address Validation is implemented in the function while updating the address state variables of the contract:

```
function setTeamAddress(address t) public restrictedOwner {  
    _teamAddress = t;  
}
```

**Resolution:** A **required** statement should be included in such functions to ensure no zero address is passed in the arguments.

**Status:** **Fixed.**

(6) Functions could be marked internal :

```
function getCountMyPartnerRewards() public view returns (uint256){  
    return _myPartnerRewards[_msgSender()].length;  
}  
  
function getCountMyStakeRewards() public view returns (uint256){  
    return _myStakeRewards[_msgSender()].length;  
}
```

The OneSeed contract includes **getCountMyPartnerRewards** and

**getCountMyStakeRewards** functions that can be marked as **internal**.

As per the current design, both of these functions simply return the length of the **\_myPartnerRewards** or **\_myStakeRewards** mapping which is used to validate the **index** argument passed in functions like **getMyPartnerReward** or **getMyStakeReward**.

Therefore, the major significance of **\_myPartnerRewards** or **\_myStakeRewards** functions lies in being called from within the contract and not from outside the contract.

Hence they could be assigned an **internal visibility**.

**Resolution:** If the visibility keyword of the above-mentioned functions is not intended, they should be marked **Internal**.

**Note:** If these functions are used on the UI side, then please ignore this.

**Status:** **Acknowledged**.

(7) External Visibility should be preferred:

Those functions that are never called throughout the contract should be marked as external visibility instead of public visibility.

Moreover, using external visibility will effectively result in Gas Optimization as well.

Therefore, the following function must be marked as external within the contract:

- `isExcludedFromReward()`
- `totalFees()`
- `deliver()`
- `reflectionFromToken()`
- `excludeFromReward()`
- `excludeFromFee()`
- `includeInFee()`
- `setSwapAndLiquifyEnabled()`
- `isExcludedFromFee()`
- `stakeReferred()`
- `lockStake()`
- `extendStakeLock()`
- `getStakingMember()`
- `setTeamAddress()`

- `getMyStakeReward()`
- `getMyPartnerReward()`
- `isAutoCompound()`
- `toggleAutoCompound()`
- `getGlobalStakingAmount()`
- `setMinimumStakingAmount()`
- `setRewardsPerCall()`
- `setGlobalMaxStakingAmount()`

**Resolution:** If the PUBLIC visibility of the above-mentioned functions is not intended, then the EXTERNAL Visibility keyword should be preferred.

**Status:** **Fixed.**

(8) Constant declaration should be preferred:

State variables that are not supposed to change throughout the contract should be declared as constant.

**Resolution:** The following state variables need to be declared as constant, unless the current contract design is intended.

- `_decimals`
- `_interestModification`
- `_liquidityReward`
- `_maxStakingAmount`
- `_minimumStakeLock`
- `_name`
- `_partnerReward`
- `_returnDivisor`
- `_rewardTime`
- `_symbol`
- `_teamReward`
- `numTokensSellToAddToLiquidity`

**Status:** **Acknowledged.**

**Very Low / Discussion / Best practices:**

(1) Coding Style Issues in the Contract :

```
Variable OneSeed._liquidityFee (contracts/OneSeed.sol#717) is not in mixedCase
Variable OneSeed._maxTxAmount (contracts/OneSeed.sol#726) is not in mixedCase
Variable OneSeed._stakingInterest (contracts/OneSeed.sol#728) is not in mixedCase
Variable OneSeed._lastStakingInterest (contracts/OneSeed.sol#729) is not in mixedCase
Variable OneSeed._interestModification (contracts/OneSeed.sol#730) is not in mixedCase
Variable OneSeed._partnerReward (contracts/OneSeed.sol#731) is not in mixedCase
Variable OneSeed._teamReward (contracts/OneSeed.sol#732) is not in mixedCase
Variable OneSeed._liquidityReward (contracts/OneSeed.sol#733) is not in mixedCase
Variable OneSeed._lastInterestAdjustment (contracts/OneSeed.sol#734) is not in mixedCase
Variable OneSeed._returnDivisor (contracts/OneSeed.sol#735) is not in mixedCase
Variable OneSeed._stakingId (contracts/OneSeed.sol#736) is not in mixedCase
Variable OneSeed._rewardTime (contracts/OneSeed.sol#737) is not in mixedCase
Variable OneSeed._teamAddress (contracts/OneSeed.sol#738) is not in mixedCase
Variable OneSeed._minimumStakeLock (contracts/OneSeed.sol#739) is not in mixedCase
Variable OneSeed._minimumStakingAmount (contracts/OneSeed.sol#740) is not in mixedCase
Variable OneSeed._maxStakingAmount (contracts/OneSeed.sol#741) is not in mixedCase
```

Code readability of a Smart Contract is largely influenced by the Coding Style issues and in some specific scenarios may lead to bugs in the future.

During the automated testing, it was found that the OneSeed contract had quite a few code style issues.

**Resolution:** Therefore, it is recommended to fix the issues like naming convention, indentation, and code layout issues in a smart contract.

**Status:** **Acknowledged.**

## Centralization

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

- **excludeFromReward:** The Owner can check if the account is already excluded.
- **excludeFromFee:** The Owner can check if the account fee is already excluded.
- **includeInReward:** The Owner can check if the account is already included or not.
- **includeInFee:** The Owner can check the included fee.
- **setTaxFeePercent:** The Owner can set the tax fee Percentage.
- **setLiquidityFeePercent:** The Owner can set Liquidity Fee Percentage.
- **setMaxTxPercent:** The Owner can set maximum Tax Percentage.
- **setSwapAndLiquifyEnabled:** The Owner can set Swap and Liquify enable status.
- **enableTrading:** The Owner can set trading enable status.

## Conclusion

We were given a contract code. And we have used all possible tests based on given objects as files. We observed some issues in the smart contracts and those are fixed/acknowledged in the smart contracts. **So it is not good to go for the production.**

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).

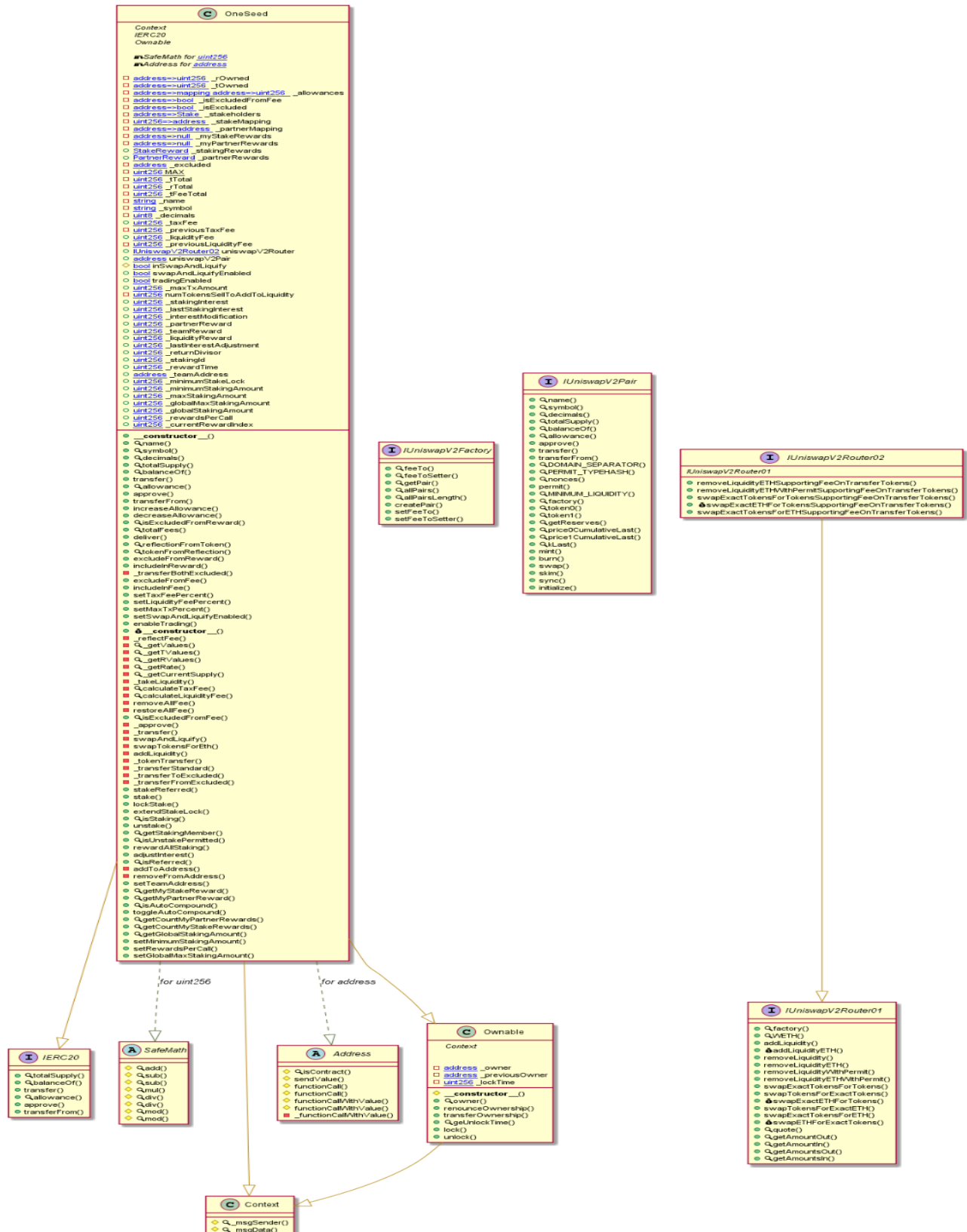
Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we 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 - OneSeed Token



This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.

Email: [audit@EtherAuthority.io](mailto:audit@EtherAuthority.io)

# Slither Results Log

## Slither log >> OneSeed.sol

INFO:Detectors:

Reentrancy in OneSeed.\_transfer(address,address,uint256) (OneSeed.sol#1047-1088):

External calls:

- swapAndLiquify(contractTokenBalance) (OneSeed.sol#1079)

- uniswapV2Router.addLiquidityETH{value:

ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (OneSeed.sol#1118-1125)

-

uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (OneSeed.sol#1107-1113)

External calls sending eth:

- swapAndLiquify(contractTokenBalance) (OneSeed.sol#1079)

- uniswapV2Router.addLiquidityETH{value:

ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (OneSeed.sol#1118-1125)

State variables written after the call(s):

- \_tokenTransfer(from,to,amount,takeFee) (OneSeed.sol#1087)

- \_rOwned[address(this)] = \_rOwned[address(this)].add(rLiquidity) (OneSeed.sol#1003)

- \_rOwned[sender] = \_rOwned[sender].sub(rAmount) (OneSeed.sol#1150)

- \_rOwned[sender] = \_rOwned[sender].sub(rAmount) (OneSeed.sol#1159)

- \_rOwned[sender] = \_rOwned[sender].sub(rAmount) (OneSeed.sol#915)

- \_rOwned[sender] = \_rOwned[sender].sub(rAmount) (OneSeed.sol#1170)

- \_rOwned[recipient] = \_rOwned[recipient].add(rTransferAmount) (OneSeed.sol#1151)

- \_rOwned[recipient] = \_rOwned[recipient].add(rTransferAmount) (OneSeed.sol#1171)

- \_rOwned[recipient] = \_rOwned[recipient].add(rTransferAmount) (OneSeed.sol#1161)

- \_rOwned[recipient] = \_rOwned[recipient].add(rTransferAmount) (OneSeed.sol#917)

- \_tokenTransfer(from,to,amount,takeFee) (OneSeed.sol#1087)

- \_rTotal = \_rTotal.sub(rFee) (OneSeed.sol#958)

- \_tokenTransfer(from,to,amount,takeFee) (OneSeed.sol#1087)

- \_tOwned[address(this)] = \_tOwned[address(this)].add(tLiquidity) (OneSeed.sol#1005)

- \_tOwned[sender] = \_tOwned[sender].sub(tAmount) (OneSeed.sol#1169)

- \_tOwned[sender] = \_tOwned[sender].sub(tAmount) (OneSeed.sol#914)

- \_tOwned[recipient] = \_tOwned[recipient].add(tTransferAmount) (OneSeed.sol#1160)

- \_tOwned[recipient] = \_tOwned[recipient].add(tTransferAmount) (OneSeed.sol#916)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities>

INFO:Detectors:

OneSeed.rewardAllStaking(uint256) (OneSeed.sol#1290-1426) performs a multiplication on the result of a division:

- nDays = diff.div(\_rewardTime) (OneSeed.sol#1326)

- reward = \_stakeholders[\_stakeMapping[i]].amount.mul(interest).mul(nDays).div(\_returnDivisor)

(OneSeed.sol#1349)

OneSeed.rewardAllStaking(uint256) (OneSeed.sol#1290-1426) performs a multiplication on the result of a division:

- reward = \_stakeholders[\_stakeMapping[i]].amount.mul(interest).mul(nDays).div(\_returnDivisor)

(OneSeed.sol#1349)

- partnerRewardAmount = reward.mul(\_partnerReward).div(\_returnDivisor) (OneSeed.sol#1369)

OneSeed.rewardAllStaking(uint256) (OneSeed.sol#1290-1426) performs a multiplication on the result of a division:

- reward = \_stakeholders[\_stakeMapping[i]].amount.mul(interest).mul(nDays).div(\_returnDivisor)

(OneSeed.sol#1349)

- tReward = reward.mul(tr).div(\_returnDivisor) (OneSeed.sol#1390)

OneSeed.rewardAllStaking(uint256) (OneSeed.sol#1290-1426) performs a multiplication on the result of a division:

- reward = \_stakeholders[\_stakeMapping[i]].amount.mul(interest).mul(nDays).div(\_returnDivisor)

(OneSeed.sol#1349)

- lReward = reward.mul(\_liquidityReward).div(\_returnDivisor) (OneSeed.sol#1394)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply>

INFO:Detectors:

OneSeed.removeFromAddress(address,uint256) (OneSeed.sol#1456-1472) uses a dangerous strict equality:

- \_rTotal == 0 (OneSeed.sol#1469)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities>

INFO:Detectors:

OneSeed.addLiquidity(uint256,uint256) (OneSeed.sol#1116-1126) ignores return value by uniswapV2Router.addLiquidityETH{value:

ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (OneSeed.sol#1118-1125)

OneSeed.rewardAllStaking(uint256) (OneSeed.sol#1290-1426) ignores return value by tr.add(\_partnerReward) (OneSeed.sol#1386)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return>

INFO:Detectors:

OneSeed.allowance(address,address).owner (OneSeed.sol#831) shadows:

- Ownable.owner() (OneSeed.sol#382-384) (function)

OneSeed.\_approve(address,address,uint256).owner (OneSeed.sol#1039) shadows:

- Ownable.owner() (OneSeed.sol#382-384) (function)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing>

INFO:Detectors:

OneSeed.setTeamAddress(address).t (OneSeed.sol#1475) lacks a zero-check on :

- \_teamAddress = t (OneSeed.sol#1476)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation>

INFO:Detectors:

Reentrancy in OneSeed.\_transfer(address,address,uint256) (OneSeed.sol#1047-1088):

External calls:

- swapAndLiquify(contractTokenBalance) (OneSeed.sol#1079)

- uniswapV2Router.addLiquidityETH{value:

ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (OneSeed.sol#1118-1125)

-

uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(t his),block.timestamp) (OneSeed.sol#1107-1113)

External calls sending eth:

- swapAndLiquify(contractTokenBalance) (OneSeed.sol#1079)

- uniswapV2Router.addLiquidityETH{value:

ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (OneSeed.sol#1118-1125)

State variables written after the call(s):

- \_tokenTransfer(from,to,amount,takeFee) (OneSeed.sol#1087)

- \_liquidityFee = \_previousLiquidityFee (OneSeed.sol#1032)

- \_liquidityFee = 0 (OneSeed.sol#1027)

- \_tokenTransfer(from,to,amount,takeFee) (OneSeed.sol#1087)

- \_previousLiquidityFee = \_liquidityFee (OneSeed.sol#1024)

- \_tokenTransfer(from,to,amount,takeFee) (OneSeed.sol#1087)

- \_previousTaxFee = \_taxFee (OneSeed.sol#1023)

- \_tokenTransfer(from,to,amount,takeFee) (OneSeed.sol#1087)

- \_tFeeTotal = \_tFeeTotal.add(tFee) (OneSeed.sol#959)

- \_tokenTransfer(from,to,amount,takeFee) (OneSeed.sol#1087)

- \_taxFee = \_previousTaxFee (OneSeed.sol#1031)

- \_taxFee = 0 (OneSeed.sol#1026)

Reentrancy in OneSeed.constructor() (OneSeed.sol#788-801):

External calls:

- uniswapV2Pair =

IUniswapV2Factory(\_uniswapV2Router.factory()).createPair(address(this),\_uniswapV2Router.WETH()) (OneSeed.sol#792-793)

State variables written after the call(s):

- \_isExcludedFromFee[owner()] = true (OneSeed.sol#795)

- \_isExcludedFromFee[address(this)] = true (OneSeed.sol#796)

- \_lastStakingInterest = \_stakingInterest (OneSeed.sol#798)

- \_teamAddress = \_msgSender() (OneSeed.sol#797)

- uniswapV2Router = \_uniswapV2Router (OneSeed.sol#794)

Reentrancy in OneSeed.swapAndLiquify(uint256) (OneSeed.sol#1090-1098):

External calls:

- swapTokensForEth(half) (OneSeed.sol#1094)

-

uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(t his),block.timestamp) (OneSeed.sol#1107-1113)

- addLiquidity(otherHalf,newBalance) (OneSeed.sol#1096)
- uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (OneSeed.sol#1118-1125)
- External calls sending eth:
  - addLiquidity(otherHalf,newBalance) (OneSeed.sol#1096)
  - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (OneSeed.sol#1118-1125)
- State variables written after the call(s):
  - addLiquidity(otherHalf,newBalance) (OneSeed.sol#1096)
  - \_allowances[owner][spender] = amount (OneSeed.sol#1043)
- Reentrancy in OneSeed.transferFrom(address,address,uint256) (OneSeed.sol#840-844):
- External calls:
  - \_transfer(sender,recipient,amount) (OneSeed.sol#841)
  - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (OneSeed.sol#1118-1125)
- 
- uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (OneSeed.sol#1107-1113)
- External calls sending eth:
  - \_transfer(sender,recipient,amount) (OneSeed.sol#841)
  - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (OneSeed.sol#1118-1125)
- State variables written after the call(s):
  - \_approve(sender,\_msgSender(),\_allowances[sender][\_msgSender()].sub(amount,ERC20: transfer amount exceeds allowance)) (OneSeed.sol#842)
  - \_allowances[owner][spender] = amount (OneSeed.sol#1043)
- Reference: <https://github.com/cryptic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2>
- INFO:Detectors:
- Reentrancy in OneSeed.\_transfer(address,address,uint256) (OneSeed.sol#1047-1088):
- External calls:
  - swapAndLiquify(contractTokenBalance) (OneSeed.sol#1079)
  - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (OneSeed.sol#1118-1125)
- 
- uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (OneSeed.sol#1107-1113)
- External calls sending eth:
  - swapAndLiquify(contractTokenBalance) (OneSeed.sol#1079)
  - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (OneSeed.sol#1118-1125)
- Event emitted after the call(s):
  - Transfer(sender,recipient,tTransferAmount) (OneSeed.sol#1154)
  - \_tokenTransfer(from,to,amount,takeFee) (OneSeed.sol#1087)
  - Transfer(sender,recipient,tTransferAmount) (OneSeed.sol#1174)
  - \_tokenTransfer(from,to,amount,takeFee) (OneSeed.sol#1087)
  - Transfer(sender,recipient,tTransferAmount) (OneSeed.sol#1164)
  - \_tokenTransfer(from,to,amount,takeFee) (OneSeed.sol#1087)
  - Transfer(sender,recipient,tTransferAmount) (OneSeed.sol#920)
  - \_tokenTransfer(from,to,amount,takeFee) (OneSeed.sol#1087)
- Reentrancy in OneSeed.constructor() (OneSeed.sol#788-801):
- External calls:
  - uniswapV2Pair = IUniswapV2Factory(\_uniswapV2Router.factory()).createPair(address(this),\_uniswapV2Router.WETH()) (OneSeed.sol#792-793)
- Event emitted after the call(s):
  - Transfer(address(0),\_msgSender(),\_tTotal) (OneSeed.sol#800)
- Reentrancy in OneSeed.swapAndLiquify(uint256) (OneSeed.sol#1090-1098):
- External calls:
  - swapTokensForEth(half) (OneSeed.sol#1094)
- 
- uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (OneSeed.sol#1107-1113)
- addLiquidity(otherHalf,newBalance) (OneSeed.sol#1096)
- uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (OneSeed.sol#1118-1125)



External calls sending eth:

- addLiquidity(otherHalf,newBalance) (OneSeed.sol#1096)
- uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (OneSeed.sol#1118-1125)

Event emitted after the call(s):

- Approval(owner,spender,amount) (OneSeed.sol#1044)
- addLiquidity(otherHalf,newBalance) (OneSeed.sol#1096)
- SwapAndLiquify(half,newBalance,otherHalf) (OneSeed.sol#1097)

Reentrancy in OneSeed.transferFrom(address,address,uint256) (OneSeed.sol#840-844):

External calls:

- \_transfer(sender,recipient,amount) (OneSeed.sol#841)
- uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (OneSeed.sol#1118-1125)

-

uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (OneSeed.sol#1107-1113)

External calls sending eth:

- \_transfer(sender,recipient,amount) (OneSeed.sol#841)
- uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (OneSeed.sol#1118-1125)

Event emitted after the call(s):

- Approval(owner,spender,amount) (OneSeed.sol#1044)
- \_approve(sender,\_msgSender(),\_allowances[sender][\_msgSender()].sub(amount,ERC20: transfer amount exceeds allowance)) (OneSeed.sol#842)

Reference: <https://github.com/cryptic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3>

INFO:Detectors:

Ownable.unlock() (OneSeed.sol#435-440) uses timestamp for comparisons

Dangerous comparisons:

- require(bool,string)(now > \_lockTime,Contract is locked) (OneSeed.sol#437)

OneSeed.reflectionFromToken(uint256,bool) (OneSeed.sol#873-882) uses timestamp for comparisons

Dangerous comparisons:

- require(bool,string)(tAmount <= \_tTotal,Amount must be less than supply) (OneSeed.sol#874)

OneSeed.tokenFromReflection(uint256) (OneSeed.sol#884-888) uses timestamp for comparisons

Dangerous comparisons:

- require(bool,string)(rAmount <= \_rTotal,Amount must be less than total reflections) (OneSeed.sol#885)

OneSeed.\_getCurrentSupply() (OneSeed.sol#988-998) uses timestamp for comparisons

Dangerous comparisons:

- \_rOwned[\_excluded[i]] > rSupply || \_tOwned[\_excluded[i]] > tSupply (OneSeed.sol#992)
- rSupply < \_rTotal.div(\_tTotal) (OneSeed.sol#996)

OneSeed.\_transfer(address,address,uint256) (OneSeed.sol#1047-1088) uses timestamp for comparisons

Dangerous comparisons:

- contractTokenBalance >= \_maxTxAmount (OneSeed.sol#1065)
- overMinTokenBalance = contractTokenBalance >= numTokensSellToAddToLiquidity (OneSeed.sol#1070)
- overMinTokenBalance && ! inSwapAndLiquify && from != uniswapV2Pair && swapAndLiquifyEnabled (OneSeed.sol#1072-1075)

OneSeed.stake(uint256) (OneSeed.sol#1184-1215) uses timestamp for comparisons

Dangerous comparisons:

- require(bool,string)(balanceOf(\_msgSender()) >= amount,low balance) (OneSeed.sol#1185)
- require(bool,string)(\_globalStakingAmount.add(amount) < \_globalMaxStakingAmount,staking pools are full) (OneSeed.sol#1188)

OneSeed.extendStakeLock(uint256) (OneSeed.sol#1229-1236) uses timestamp for comparisons

Dangerous comparisons:

- require(bool,string)(\_stakeholders[\_msgSender()].lockedUntil < now,expired lock) (OneSeed.sol#1232)

OneSeed.isUnstakePermitted(address) (OneSeed.sol#1281-1288) uses timestamp for comparisons

Dangerous comparisons:

- ! isStaking(a) || \_stakeholders[a].amount == 0 || (\_stakeholders[a].locked && now < \_stakeholders[a].lockedUntil) (OneSeed.sol#1283)

OneSeed.rewardAllStaking(uint256) (OneSeed.sol#1290-1426) uses timestamp for comparisons

Dangerous comparisons:

- nDays >= 1 (OneSeed.sol#1330)
- \_stakeholders[\_stakeMapping[i]].lockedUntil > now (OneSeed.sol#1337)

- `_stakeholders[_stakeMapping[i]].autoCompoundOn &&`
- `_stakeholders[_stakeMapping[i]].amount.add(reward) <= _maxStakingAmount &&`
- `_globalStakingAmount.add(reward) <= _globalMaxStakingAmount (OneSeed.sol#1354-1356)`
- `OneSeed.adjustInterest() (OneSeed.sol#1429-1436)` uses timestamp for comparisons
- Dangerous comparisons:
  - `now.sub(_lastInterestAdjustment) >= _rewardTime (OneSeed.sol#1430)`
- `OneSeed.removeFromAddress(address,uint256) (OneSeed.sol#1456-1472)` uses timestamp for comparisons
- Dangerous comparisons:
  - `_tTotal == 0 (OneSeed.sol#1465)`
  - `_rTotal == 0 (OneSeed.sol#1469)`

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp>

INFO:Detectors:

`Address.isContract(address) (OneSeed.sol#246-255)` uses assembly

- `INLINE ASM (OneSeed.sol#253)`

`Address._functionCallWithValue(address,bytes,uint256,string) (OneSeed.sol#339-360)` uses assembly

- `INLINE ASM (OneSeed.sol#352-355)`

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage>

INFO:Detectors:

`OneSeed.rewardAllStaking(uint256) (OneSeed.sol#1290-1426)` has costly operations inside a loop:

- `_globalStakingAmount = _globalStakingAmount.add(reward) (OneSeed.sol#1359)`

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#costly-operations-inside-a-loop>

INFO:Detectors:

`Address._functionCallWithValue(address,bytes,uint256,string) (OneSeed.sol#339-360)` is never used and should be removed

`Address.functionCall(address,bytes) (OneSeed.sol#299-301)` is never used and should be removed

`Address.functionCall(address,bytes,string) (OneSeed.sol#309-311)` is never used and should be removed

`Address.functionCallWithValue(address,bytes,uint256) (OneSeed.sol#324-326)` is never used and should be removed

`Address.functionCallWithValue(address,bytes,uint256,string) (OneSeed.sol#334-337)` is never used and should be removed

`Address.isContract(address) (OneSeed.sol#246-255)` is never used and should be removed

`Address.sendValue(address,uint256) (OneSeed.sol#273-279)` is never used and should be removed

`Context._msgData() (OneSeed.sol#221-225)` is never used and should be removed

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code>

INFO:Detectors:

`OneSeed._previousTaxFee (OneSeed.sol#716)` is set pre-construction with a non-constant function or state variable:

- `_taxFee`

`OneSeed._previousLiquidityFee (OneSeed.sol#718)` is set pre-construction with a non-constant function or state variable:

- `_liquidityFee`

`OneSeed._currentRewardIndex (OneSeed.sol#745)` is set pre-construction with a non-constant function or state variable:

- `_stakingId`

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#function-initializing-state-variables>

INFO:Detectors:

Low level call in `Address.sendValue(address,uint256) (OneSeed.sol#273-279)`:

- `(success) = recipient.call{value: amount}() (OneSeed.sol#277)`

Low level call in `Address._functionCallWithValue(address,bytes,uint256,string) (OneSeed.sol#339-360)`:

- `(success,returndata) = target.call{value: weiValue}(data) (OneSeed.sol#343)`

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls>

INFO:Detectors:

Function `IUniswapV2Pair.DOMAIN_SEPARATOR() (OneSeed.sol#485)` is not in mixedCase

Function `IUniswapV2Pair.PERMIT_TYPEHASH() (OneSeed.sol#487)` is not in mixedCase

Function `IUniswapV2Pair.MINIMUM_LIQUIDITY() (OneSeed.sol#505)` is not in mixedCase

Function `IUniswapV2Router01.WETH() (OneSeed.sol#537)` is not in mixedCase

Parameter `OneSeed.setSwapAndLiquifyEnabled(bool)._enabled (OneSeed.sol#945)` is not in mixedCase

Parameter `OneSeed.calculateTaxFee(uint256)._amount (OneSeed.sol#1008)` is not in mixedCase

Parameter `OneSeed.calculateLiquidityFee(uint256)._amount (OneSeed.sol#1014)` is not in mixedCase

Variable `OneSeed._stakingRewards (OneSeed.sol#704)` is not in mixedCase

Variable `OneSeed._partnerRewards (OneSeed.sol#705)` is not in mixedCase

Variable `OneSeed._taxFee (OneSeed.sol#715)` is not in mixedCase

Variable `OneSeed._liquidityFee (OneSeed.sol#717)` is not in mixedCase

Variable `OneSeed._maxTxAmount (OneSeed.sol#726)` is not in mixedCase



Variable OneSeed.\_stakingInterest (OneSeed.sol#728) is not in mixedCase  
Variable OneSeed.\_lastStakingInterest (OneSeed.sol#729) is not in mixedCase  
Variable OneSeed.\_interestModification (OneSeed.sol#730) is not in mixedCase  
Variable OneSeed.\_partnerReward (OneSeed.sol#731) is not in mixedCase  
Variable OneSeed.\_teamReward (OneSeed.sol#732) is not in mixedCase  
Variable OneSeed.\_liquidityReward (OneSeed.sol#733) is not in mixedCase  
Variable OneSeed.\_lastInterestAdjustment (OneSeed.sol#734) is not in mixedCase  
Variable OneSeed.\_returnDivisor (OneSeed.sol#735) is not in mixedCase  
Variable OneSeed.\_stakingId (OneSeed.sol#736) is not in mixedCase  
Variable OneSeed.\_rewardTime (OneSeed.sol#737) is not in mixedCase  
Variable OneSeed.\_teamAddress (OneSeed.sol#738) is not in mixedCase  
Variable OneSeed.\_minimumStakeLock (OneSeed.sol#739) is not in mixedCase  
Variable OneSeed.\_minimumStakingAmount (OneSeed.sol#740) is not in mixedCase  
Variable OneSeed.\_maxStakingAmount (OneSeed.sol#741) is not in mixedCase  
Variable OneSeed.\_globalMaxStakingAmount (OneSeed.sol#742) is not in mixedCase  
Variable OneSeed.\_globalStakingAmount (OneSeed.sol#743) is not in mixedCase  
Variable OneSeed.\_rewardsPerCall (OneSeed.sol#744) is not in mixedCase  
Variable OneSeed.\_currentRewardIndex (OneSeed.sol#745) is not in mixedCase  
Modifier OneSeed.OnlyActiveStaker() (OneSeed.sol#1543-1548) is not in mixedCase

Reference:

<https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions>

INFO:Detectors:

Redundant expression "this (OneSeed.sol#222)" inContext (OneSeed.sol#216-226)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements>

INFO:Detectors:

Variable

IUniswapV2Router01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amountADesired (OneSeed.sol#542) is too similar to

IUniswapV2Router01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amountBDesired (OneSeed.sol#543)

Variable OneSeed.\_transferBothExcluded(address,address,uint256).rTransferAmount (OneSeed.sol#913) is too similar to OneSeed.\_transferBothExcluded(address,address,uint256).tTransferAmount (OneSeed.sol#913)

Variable OneSeed.\_transferFromExcluded(address,address,uint256).rTransferAmount (OneSeed.sol#1168) is too similar to OneSeed.\_transferFromExcluded(address,address,uint256).tTransferAmount (OneSeed.sol#1168)

Variable OneSeed.\_transferStandard(address,address,uint256).rTransferAmount (OneSeed.sol#1149) is too similar to OneSeed.\_getValues(uint256).tTransferAmount (OneSeed.sol#963)

Variable OneSeed.\_transferStandard(address,address,uint256).rTransferAmount (OneSeed.sol#1149) is too similar to OneSeed.\_transferToExcluded(address,address,uint256).tTransferAmount (OneSeed.sol#1158)

Variable OneSeed.reflectionFromToken(uint256,bool).rTransferAmount (OneSeed.sol#879) is too similar to OneSeed.\_transferFromExcluded(address,address,uint256).tTransferAmount (OneSeed.sol#1168)

Variable OneSeed.\_transferFromExcluded(address,address,uint256).rTransferAmount (OneSeed.sol#1168) is too similar to OneSeed.\_transferBothExcluded(address,address,uint256).tTransferAmount (OneSeed.sol#913)

Variable OneSeed.reflectionFromToken(uint256,bool).rTransferAmount (OneSeed.sol#879) is too similar to OneSeed.\_getValues(uint256).tTransferAmount (OneSeed.sol#963)

Variable OneSeed.reflectionFromToken(uint256,bool).rTransferAmount (OneSeed.sol#879) is too similar to OneSeed.\_transferToExcluded(address,address,uint256).tTransferAmount (OneSeed.sol#1158)

Variable OneSeed.\_getValues(uint256).rTransferAmount (OneSeed.sol#964) is too similar to OneSeed.\_transferToExcluded(address,address,uint256).tTransferAmount (OneSeed.sol#1158)

Variable OneSeed.\_getValues(uint256).rTransferAmount (OneSeed.sol#964) is too similar to OneSeed.\_getValues(uint256).tTransferAmount (OneSeed.sol#963)

Variable OneSeed.\_getValues(uint256).rTransferAmount (OneSeed.sol#964) is too similar to OneSeed.\_transferFromExcluded(address,address,uint256).tTransferAmount (OneSeed.sol#1168)

Variable OneSeed.\_transferFromExcluded(address,address,uint256).rTransferAmount (OneSeed.sol#1168) is too similar to OneSeed.\_getTValues(uint256).tTransferAmount (OneSeed.sol#971)

Variable OneSeed.\_transferStandard(address,address,uint256).rTransferAmount (OneSeed.sol#1149) is too similar to OneSeed.\_transferFromExcluded(address,address,uint256).tTransferAmount (OneSeed.sol#1168)

Variable OneSeed.\_transferStandard(address,address,uint256).rTransferAmount (OneSeed.sol#1149) is too similar to OneSeed.\_transferBothExcluded(address,address,uint256).tTransferAmount (OneSeed.sol#913)

Variable OneSeed.\_getValues(uint256).rTransferAmount (OneSeed.sol#964) is too similar to OneSeed.\_getTValues(uint256).tTransferAmount (OneSeed.sol#971)

Variable OneSeed.\_getValues(uint256).rTransferAmount (OneSeed.sol#964) is too similar to OneSeed.\_transferStandard(address,address,uint256).tTransferAmount (OneSeed.sol#1149)

Variable OneSeed.\_getValues(uint256).rTransferAmount (OneSeed.sol#964) is too similar to OneSeed.\_transferBothExcluded(address,address,uint256).tTransferAmount (OneSeed.sol#913)

Variable OneSeed.\_transferStandard(address,address,uint256).rTransferAmount (OneSeed.sol#1149) is too similar to OneSeed.\_getTValues(uint256).tTransferAmount (OneSeed.sol#971)

Variable OneSeed.\_transferToExcluded(address,address,uint256).rTransferAmount (OneSeed.sol#1158) is too similar to OneSeed.\_getValues(uint256).tTransferAmount (OneSeed.sol#963)

Variable OneSeed.\_transferToExcluded(address,address,uint256).rTransferAmount (OneSeed.sol#1158) is too similar to OneSeed.\_transferToExcluded(address,address,uint256).tTransferAmount (OneSeed.sol#1158)

Variable OneSeed.\_transferStandard(address,address,uint256).rTransferAmount (OneSeed.sol#1149) is too similar to OneSeed.\_transferStandard(address,address,uint256).tTransferAmount (OneSeed.sol#1149)

Variable OneSeed.\_getRValues(uint256,uint256,uint256,uint256).rTransferAmount (OneSeed.sol#979) is too similar to OneSeed.\_getValues(uint256).tTransferAmount (OneSeed.sol#963)

Variable OneSeed.\_getRValues(uint256,uint256,uint256,uint256).rTransferAmount (OneSeed.sol#979) is too similar to OneSeed.\_transferToExcluded(address,address,uint256).tTransferAmount (OneSeed.sol#1158)

Variable OneSeed.\_transferBothExcluded(address,address,uint256).rTransferAmount (OneSeed.sol#913) is too similar to OneSeed.\_getValues(uint256).tTransferAmount (OneSeed.sol#963)

Variable OneSeed.\_transferBothExcluded(address,address,uint256).rTransferAmount (OneSeed.sol#913) is too similar to OneSeed.\_transferToExcluded(address,address,uint256).tTransferAmount (OneSeed.sol#1158)

Variable OneSeed.\_transferBothExcluded(address,address,uint256).rTransferAmount (OneSeed.sol#913) is too similar to OneSeed.\_getTValues(uint256).tTransferAmount (OneSeed.sol#971)

Variable OneSeed.\_transferFromExcluded(address,address,uint256).rTransferAmount (OneSeed.sol#1168) is too similar to OneSeed.\_getValues(uint256).tTransferAmount (OneSeed.sol#963)

Variable OneSeed.\_transferFromExcluded(address,address,uint256).rTransferAmount (OneSeed.sol#1168) is too similar to OneSeed.\_transferToExcluded(address,address,uint256).tTransferAmount (OneSeed.sol#1158)

Variable OneSeed.reflectionFromToken(uint256,bool).rTransferAmount (OneSeed.sol#879) is too similar to OneSeed.\_transferBothExcluded(address,address,uint256).tTransferAmount (OneSeed.sol#913)

Variable OneSeed.reflectionFromToken(uint256,bool).rTransferAmount (OneSeed.sol#879) is too similar to OneSeed.\_getTValues(uint256).tTransferAmount (OneSeed.sol#971)

Variable OneSeed.\_transferToExcluded(address,address,uint256).rTransferAmount (OneSeed.sol#1158) is too similar to OneSeed.\_getTValues(uint256).tTransferAmount (OneSeed.sol#971)

Variable OneSeed.\_transferBothExcluded(address,address,uint256).rTransferAmount (OneSeed.sol#913) is too similar to OneSeed.\_transferFromExcluded(address,address,uint256).tTransferAmount (OneSeed.sol#1168)

Variable OneSeed.\_transferToExcluded(address,address,uint256).rTransferAmount (OneSeed.sol#1158) is too similar to OneSeed.\_transferBothExcluded(address,address,uint256).tTransferAmount (OneSeed.sol#913)

Variable OneSeed.\_getRValues(uint256,uint256,uint256,uint256).rTransferAmount (OneSeed.sol#979) is too similar to OneSeed.\_transferBothExcluded(address,address,uint256).tTransferAmount (OneSeed.sol#913)

Variable OneSeed.reflectionFromToken(uint256,bool).rTransferAmount (OneSeed.sol#879) is too similar to OneSeed.\_transferStandard(address,address,uint256).tTransferAmount (OneSeed.sol#1149)

Variable OneSeed.\_getRValues(uint256,uint256,uint256,uint256).rTransferAmount (OneSeed.sol#979) is too similar to OneSeed.\_getTValues(uint256).tTransferAmount (OneSeed.sol#971)

Variable OneSeed.\_getRValues(uint256,uint256,uint256,uint256).rTransferAmount (OneSeed.sol#979) is too similar to OneSeed.\_transferFromExcluded(address,address,uint256).tTransferAmount (OneSeed.sol#1168)

Variable OneSeed.\_transferToExcluded(address,address,uint256).rTransferAmount (OneSeed.sol#1158) is too similar to OneSeed.\_transferStandard(address,address,uint256).tTransferAmount (OneSeed.sol#1149)

Variable OneSeed.\_getRValues(uint256,uint256,uint256,uint256).rTransferAmount (OneSeed.sol#979) is too similar to OneSeed.\_transferStandard(address,address,uint256).tTransferAmount (OneSeed.sol#1149)

Variable OneSeed.\_transferFromExcluded(address,address,uint256).rTransferAmount (OneSeed.sol#1168) is too similar to OneSeed.\_transferStandard(address,address,uint256).tTransferAmount (OneSeed.sol#1149)

Variable OneSeed.\_transferToExcluded(address,address,uint256).rTransferAmount (OneSeed.sol#1158) is too similar to OneSeed.\_transferFromExcluded(address,address,uint256).tTransferAmount (OneSeed.sol#1168)

Variable OneSeed.\_transferBothExcluded(address,address,uint256).rTransferAmount (OneSeed.sol#913) is too similar to OneSeed.\_transferStandard(address,address,uint256).tTransferAmount (OneSeed.sol#1149)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-are-too-similar>

INFO:Detectors:

OneSeed.slitherConstructorVariables() (OneSeed.sol#688-1551) uses literals with too many digits:

- numTokensSellToAddToLiquidity = 5000000 (OneSeed.sol#727)

OneSeed.slitherConstructorVariables() (OneSeed.sol#688-1551) uses literals with too many digits:

- \_interestModification = 100000 (OneSeed.sol#730)

OneSeed.slitherConstructorVariables() (OneSeed.sol#688-1551) uses literals with too many digits:

- `_partnerReward = 5000000` (OneSeed.sol#731)

OneSeed.slitherConstructorVariables() (OneSeed.sol#688-1551) uses literals with too many digits:

- `_teamReward = 2500000` (OneSeed.sol#732)

OneSeed.slitherConstructorVariables() (OneSeed.sol#688-1551) uses literals with too many digits:

- `_liquidityReward = 2500000` (OneSeed.sol#733)

OneSeed.slitherConstructorVariables() (OneSeed.sol#688-1551) uses literals with too many digits:

- `_returnDivisor = 100000000` (OneSeed.sol#735)

OneSeed.slitherConstructorVariables() (OneSeed.sol#688-1551) uses literals with too many digits:

- `_maxStakingAmount = 50000000` (OneSeed.sol#741)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits>

INFO:Detectors:

OneSeed.MAX (OneSeed.sol#708) is never used in OneSeed (OneSeed.sol#688-1551)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variables>

INFO:Detectors:

OneSeed.\_decimals (OneSeed.sol#714) should be constant

OneSeed.\_interestModification (OneSeed.sol#730) should be constant

OneSeed.\_liquidityReward (OneSeed.sol#733) should be constant

OneSeed.\_maxStakingAmount (OneSeed.sol#741) should be constant

OneSeed.\_minimumStakeLock (OneSeed.sol#739) should be constant

OneSeed.\_name (OneSeed.sol#712) should be constant

OneSeed.\_partnerReward (OneSeed.sol#731) should be constant

OneSeed.\_returnDivisor (OneSeed.sol#735) should be constant

OneSeed.\_rewardTime (OneSeed.sol#737) should be constant

OneSeed.\_symbol (OneSeed.sol#713) should be constant

OneSeed.\_teamReward (OneSeed.sol#732) should be constant

OneSeed.numTokensSellToAddToLiquidity (OneSeed.sol#727) should be constant

Reference:

<https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant>

INFO:Detectors:

renounceOwnership() should be declared external:

- Ownable.renounceOwnership() (OneSeed.sol#407-410)

transferOwnership(address) should be declared external:

- Ownable.transferOwnership(address) (OneSeed.sol#416-420)

geUnlockTime() should be declared external:

- Ownable.geUnlockTime() (OneSeed.sol#422-424)

lock(uint256) should be declared external:

- Ownable.lock(uint256) (OneSeed.sol#427-432)

unlock() should be declared external:

- Ownable.unlock() (OneSeed.sol#435-440)

name() should be declared external:

- OneSeed.name() (OneSeed.sol#805-807)

symbol() should be declared external:

- OneSeed.symbol() (OneSeed.sol#809-811)

decimals() should be declared external:

- OneSeed.decimals() (OneSeed.sol#813-815)

totalSupply() should be declared external:

- OneSeed.totalSupply() (OneSeed.sol#817-819)

transfer(address,uint256) should be declared external:

- OneSeed.transfer(address,uint256) (OneSeed.sol#826-829)

allowance(address,address) should be declared external:

- OneSeed.allowance(address,address) (OneSeed.sol#831-833)

approve(address,uint256) should be declared external:

- OneSeed.approve(address,uint256) (OneSeed.sol#835-838)

transferFrom(address,address,uint256) should be declared external:

- OneSeed.transferFrom(address,address,uint256) (OneSeed.sol#840-844)

increaseAllowance(address,uint256) should be declared external:

- OneSeed.increaseAllowance(address,uint256) (OneSeed.sol#846-849)

decreaseAllowance(address,uint256) should be declared external:

- OneSeed.decreaseAllowance(address,uint256) (OneSeed.sol#851-854)

isExcludedFromReward(address) should be declared external:

- OneSeed.isExcludedFromReward(address) (OneSeed.sol#856-858)

totalFees() should be declared external:

- OneSeed.totalFees() (OneSeed.sol#860-862)

deliver(uint256) should be declared external:

- OneSeed.deliver(uint256) (OneSeed.sol#864-871)

reflectionFromToken(uint256,bool) should be declared external:

- OneSeed.reflectionFromToken(uint256,bool) (OneSeed.sol#873-882)

excludeFromReward(address) should be declared external:

- OneSeed.excludeFromReward(address) (OneSeed.sol#890-897)

excludeFromFee(address) should be declared external:

- OneSeed.excludeFromFee(address) (OneSeed.sol#923-925)

includeInFee(address) should be declared external:

- OneSeed.includeInFee(address) (OneSeed.sol#927-929)

setSwapAndLiquifyEnabled(bool) should be declared external:

- OneSeed.setSwapAndLiquifyEnabled(bool) (OneSeed.sol#945-948)

isExcludedFromFee(address) should be declared external:

- OneSeed.isExcludedFromFee(address) (OneSeed.sol#1035-1037)

stakeReferred(uint256,address) should be declared external:

- OneSeed.stakeReferred(uint256,address) (OneSeed.sol#1177-1182)

lockStake(uint256) should be declared external:

- OneSeed.lockStake(uint256) (OneSeed.sol#1218-1226)

extendStakeLock(uint256) should be declared external:

- OneSeed.extendStakeLock(uint256) (OneSeed.sol#1229-1236)

getStakingMember(address) should be declared external:

- OneSeed.getStakingMember(address) (OneSeed.sol#1268-1278)

setTeamAddress(address) should be declared external:

- OneSeed.setTeamAddress(address) (OneSeed.sol#1475-1477)

getMyStakeReward(uint256) should be declared external:

- OneSeed.getMyStakeReward(uint256) (OneSeed.sol#1480-1491)

getMyPartnerReward(uint256) should be declared external:

- OneSeed.getMyPartnerReward(uint256) (OneSeed.sol#1494-1505)

isAutoCompound() should be declared external:

- OneSeed.isAutoCompound() (OneSeed.sol#1507-1509)

toggleAutoCompound() should be declared external:

- OneSeed.toggleAutoCompound() (OneSeed.sol#1511-1513)

getGlobalStakingAmount() should be declared external:

- OneSeed.getGlobalStakingAmount() (OneSeed.sol#1523-1525)

setMinimumStakingAmount(uint256) should be declared external:

- OneSeed.setMinimumStakingAmount(uint256) (OneSeed.sol#1527-1530)

setRewardsPerCall(uint256) should be declared external:

- OneSeed.setRewardsPerCall(uint256) (OneSeed.sol#1532-1535)

setGlobalMaxStakingAmount(uint256) should be declared external:

- OneSeed.setGlobalMaxStakingAmount(uint256) (OneSeed.sol#1537-1541)

Reference:

<https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external>

INFO:Slither:OneSeed.sol analyzed (10 contracts with 75 detectors), 178 result(s) found

INFO:Slither:Use <https://crytic.io/> to get access to additional detectors and Github integration





This is a private and confidential document. No part of this document should be disclosed to third party without prior written permission of EtherAuthority.

**Email: [audit@EtherAuthority.io](mailto:audit@EtherAuthority.io)**