February 2023

Commit 670f48222bc71c2829db29f0b87999a40255f4e1

Audited by © cyberscope

## Table of Contents

[**Table of Contents**](#_jqkdq0pni70) **1**

[**Review**](#_30j0zll) **5**

[**Audit Updates**](#_v3sar6sdrddl) **5**

[**Source Files**](#_kasq2rdywsd9) **5**

[**Introduction**](#_ja7j5t1q4u34) **6**

[**Vesting Formula**](#_hph8iq79k92u) **7**

[**DevContract**](#_nr94rh3beblq) **7**

[**INV\_Seed**](#_vc13a8heqz3w) **8**

[**INV\_Round1**](#_j06kyx4c4gnx) **8**

[**INV\_Round2**](#_f7e4av80u7ss) **9**

[**SeedReplica**](#_db8uyeb61mrl) **9**

[**Roles**](#_67me1qswnqqj) **10**

[**Owner**](#_kt3ahg6fahub) **10**

[**Operator**](#_ipvx9wr7v29i) **10**

[**User**](#_6fvdier8l2jy) **10**

[**Diagnostics**](#_wlb02ptmin4i) **11**

[**CSC - Contract State Configuration**](#_kutxro4c6z57) **13**

[**Description**](#_u5vqmasil96b) **13**

[**Recommendation**](#_8iabjvhju1x2) **13**

[**SOIL - Stable Operations in Loop**](#_oxq68yf78t20) **14**

[**Description**](#_czamohbjvtui) **14**

[**Recommendation**](#_pulq3qqwcnf6) **14**

[**GCC - Gas Consuming Calculations**](#_xqeqmy5bk7u6) **15**

[**Description**](#_kgs4no8ke8fn) **15**

[**Recommendation**](#_2628vnupuf6b) **15**

[**MC - Missing Check**](#_qgsatapjplyf) **16**

[**Description**](#_kxxw76ursl72) **16**

[**Recommendation**](#_qavep0z9c2zn) **16**

[**CR - Calculation Repetition**](#_95gel4cazw25) **17**

[**Description**](#_vrkvn322jj3k) **17**

[**Recommendation**](#_xwak7grt212v) **17**

[**PTAI - Potential Transfer Amount Inconsistency**](#_lf7jw47gbov6) **18**

[**Description**](#_o3fu8gx5vlpt) **18**

[**Recommendation**](#_7fmx0luy3q31) **19**

[**RSML - Redundant SafeMath Library**](#_sl0f7akkqjbg) **20**

[**Description**](#_sut3yfqfvtbj) **20**

[**Recommendation**](#_xgylw21meob2) **20**

[**OCTD - Transfers Contract's Tokens**](#_xqm6e32qtqu8) **21**

[**Description**](#_8wg67mm5h7nn) **21**

[**Recommendation**](#_yxj0muoshafa) **21**

[**UVF - Unused Variables And Functions**](#_b5zmx7e11jrr) **22**

[**Description**](#_itc28wxw5ghz) **22**

[**Recommendation**](#_c2ss8mebeiae) **22**

[**UF - Unimplemented Function**](#_bpijeeub8f23) **23**

[**Description**](#_dovpdqdbapf9) **23**

[**Recommendation**](#_x3lqf5rakzom) **23**

[**CR - Code Repetition**](#_k182g8xu6wl0) **24**

[**Description**](#_x6x00u55clwt) **24**

[**Recommendation**](#_p0n830ou5bli) **24**

[**ZD - Zero Division**](#_qcpmco2y0iq7) **25**

[**Description**](#_so6kl05f6h49) **25**

[**Recommendation**](#_oqk4v36zq1di) **26**

[**CO - Code Optimization**](#_9m9447ygfh3n) **27**

[**Description**](#_1gyr1hsy5w5b) **27**

[**Recommendation**](#_p17anweih14) **27**

[**RVA - Redundant Variable Assignment**](#_grx5i4m6cs3f) **28**

[**Description**](#_lm6pbi5gp33w) **28**

[**Recommendation**](#_2qhbi5xmymu1) **28**

[**DPI - Decimals Precision Inconsistency**](#_b85ds21g6h88) **29**

[**Description**](#_aa3t8ixr818z) **29**

[**Recommendation**](#_b6duezca05cq) **30**

[**TUU - Time Units Usage**](#_dstf3ttpvuvj) **31**

[**Description**](#_vc2sg68gbca0) **31**

[**Recommendation**](#_n5w9t5luxfq8) **31**

[**AAO - Accumulated Amount Overflow**](#_hur9exfzz91l) **32**

[**Description**](#_q3l02lpsug0u) **32**

[**Recommendation**](#_yjo2zyxeilq) **32**

[**L02 - State Variables could be Declared Constant**](#_dil1scrvi91j) **33**

[**Description**](#_9nuekerycmul) **33**

[**Recommendation**](#_gtqbdzrns1wd) **33**

[**L04 - Conformance to Solidity Naming Conventions**](#_iwu8rox5tlqi) **34**

[**Description**](#_2bpxutc12ber) **34**

[**Recommendation**](#_5ar9hh5zsn4a) **35**

[**L05 - Unused State Variable**](#_2hnpou40sqmi) **36**

[**Description**](#_h5ssroj4hngj) **36**

[**Recommendation**](#_j0qyehb7b3j) **36**

[**L07 - Missing Events Arithmetic**](#_yqpt43t6e0nb) **37**

[**Description**](#_1ehrlcjygddl) **37**

[**Recommendation**](#_j2oelax4q77r) **37**

[**L09 - Dead Code Elimination**](#_wz3dqw9gzfs8) **38**

[**Description**](#_p8o5oubco3tb) **38**

[**Recommendation**](#_2zivufcdt0i0) **39**

[**L12 - Using Variables before Declaration**](#_5cxg3ck5la8v) **40**

[**Description**](#_r7u081r0k1np) **40**

[**Recommendation**](#_gpnpgsx0hw2t) **40**

[**L13 - Divide before Multiply Operation**](#_31l9huyw4ik1) **41**

[**Description**](#_uydgpvazd6s8) **41**

[**Recommendation**](#_vb4cyrnmq7lc) **41**

[**L14 - Uninitialized Variables in Local Scope**](#_zg5trzg7pipb) **42**

[**Description**](#_od571yvuogjv) **42**

[**Recommendation**](#_fi3xtjrdqjeh) **42**

[**L16 - Validate Variable Setters**](#_u6q6qphqmmj9) **43**

[**Description**](#_buwowxv4573x) **43**

[**Recommendation**](#_l190xc6xwucy) **43**

[**L17 - Usage of Solidity Assembly**](#_6kmv6g8h33f) **44**

[**Description**](#_h3qzd3r1mres) **44**

[**Recommendation**](#_1ziuslh6xhl6) **44**

[**L19 - Stable Compiler Version**](#_goc6an3xv1sy) **45**

[**Description**](#_13pne9wuwnwp) **45**

[**Recommendation**](#_azhd9ycutg5d) **45**

[**L20 - Succeeded Transfer Check**](#_7xwxko4v8lo8) **46**

[**Description**](#_8jboxhie2n6) **46**

[**Recommendation**](#_eiqbgkz68yaf) **46**

[**Functions Analysis**](#_2s8eyo1) **47**

[**Inheritance Graph**](#_1px4chw6c8ix) **56**

[**Flow Graph**](#_yqe80mnqy9sz) **57**

[**Summary**](#_r68j2xczzx14) **58**

[**Disclaimer**](#_35nkun2) **59**

[**About Cyberscope**](#_1ksv4uv) **60**

## Review

| **Repository** | https://github.com/ammagtech/OGB-ICO-SmartContract |
| --- | --- |
| **Commit** | 670f48222bc71c2829db29f0b87999a40255f4e1 |

### Audit Updates

| **Initial Audit** | 13 Feb 2023 |
| --- | --- |

### Source Files

| **Filename** | SHA256 |
| --- | --- |
| **Console.sol** | 1982f3779eeec3143b12365735adb00218c417603b7c22870a8ec749d225abb8 |
| **DevContract.sol** | 54a933da8b206c020a824371fda1db8d7d5a45d50170adb34bbaa5b6756289a8 |
| **INV\_Round1.sol** | 47514cf13b128c684e9b272d2805b1d6613dbdbf2af25f99703adae443f13862 |
| **INV\_Round2.sol** | ba672130a9ae7525ecd7674b56a37006f32044d2d79b2f9fde24277b869b166d |
| **INV\_Seed.sol** | 43cca8b6264a43aefafdc45e800f86380cfb8923eec220da68ca4d88001d7b9c |
| **LaunchContract.sol** | f8f512429bbfa319ca70def64c57359a2200480d1e9be584d8298c9ab602f459 |
| **SeedReplica.sol** | b3b305e7959cabb4187c6ca7d1bb8f3fd019e705288b681dfe8a2ad7380749f7 |
| **USDToken.sol** | 56e2439c286ac369377192a9668759e869241c5d718fa8e321cbd20e483345f7 |

## Introduction

The Open Games Builders (OGB) ecosystem consists of 9 smart contracts.

1. **OGBToken** - An ERC20 token.
2. **USDToken** - An ERC20 token.
3. **LaunchContract** - A utility contract to launch the INV\_Seed and INV\_Round1 contracts.
4. **DevContract** - A vesting contract.
5. **INV\_Seed** - A vesting contract.
6. **INV\_Round1** - A vesting contract.
7. **INV\_Round2** - A vesting contract.
8. **SeedReplica** - A vesting contract for users that have prepaid.
9. **Console** - A library that handles logging.

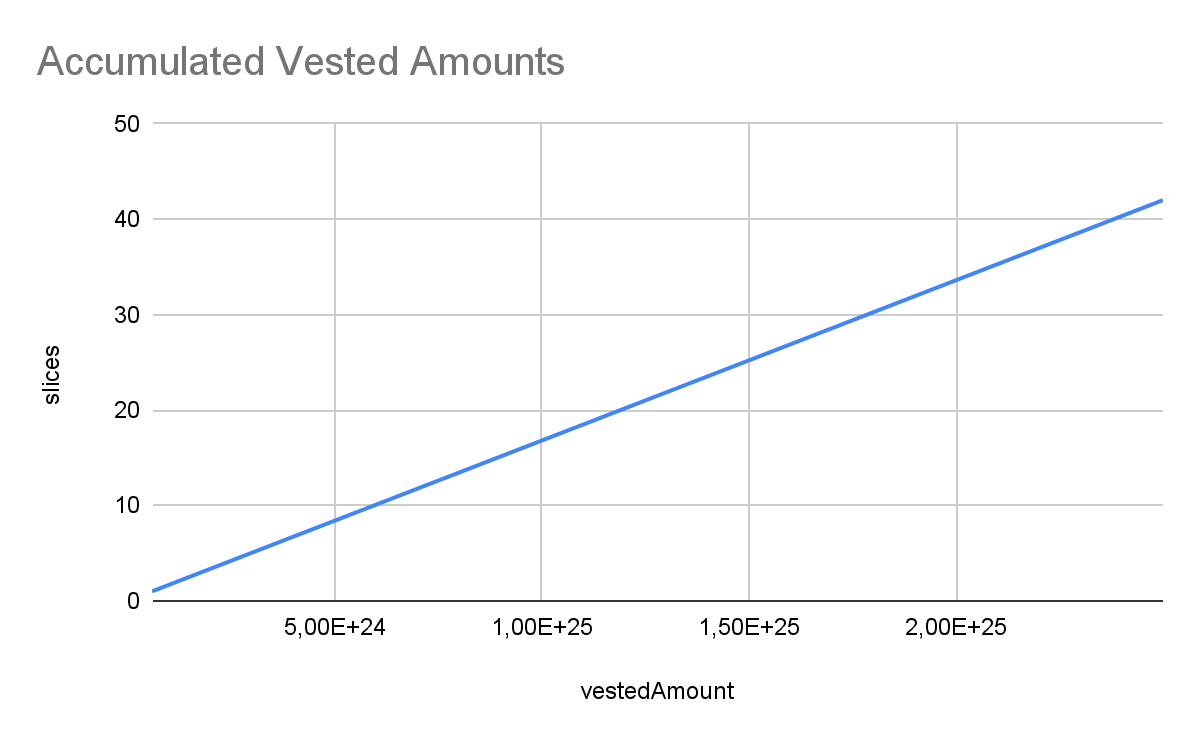
The purpose of the ecosystem is to handle token sales separated into rounds, where investors can deposit USDT and receive OGB tokens in return. The contracts keep track of how much USDT each investor has deposited, how many OGB tokens they have received, and how many tokens they are entitled to after a vesting period. The contracts also include functionality for controlling the start and stop of the ICO (Initial Coin Offering), setting the USDT to OGB rate, and various other parameters.

It should be noted that all of the vesting contracts inherit the functionality of the UUPSUpgradable contract. This allows the creation of an upgradeable smart contract that can be updated with bug fixes, security patches, or new features without having to deploy a completely new contract and update the addresses in every application that interacts with it. This makes the upgrade process much easier, cheaper, and more efficient for the project and its users. You can read more about it at <https://docs.openzeppelin.com/contracts/4.x/api/proxy#UUPSUpgradeable>.

### Vesting Formula

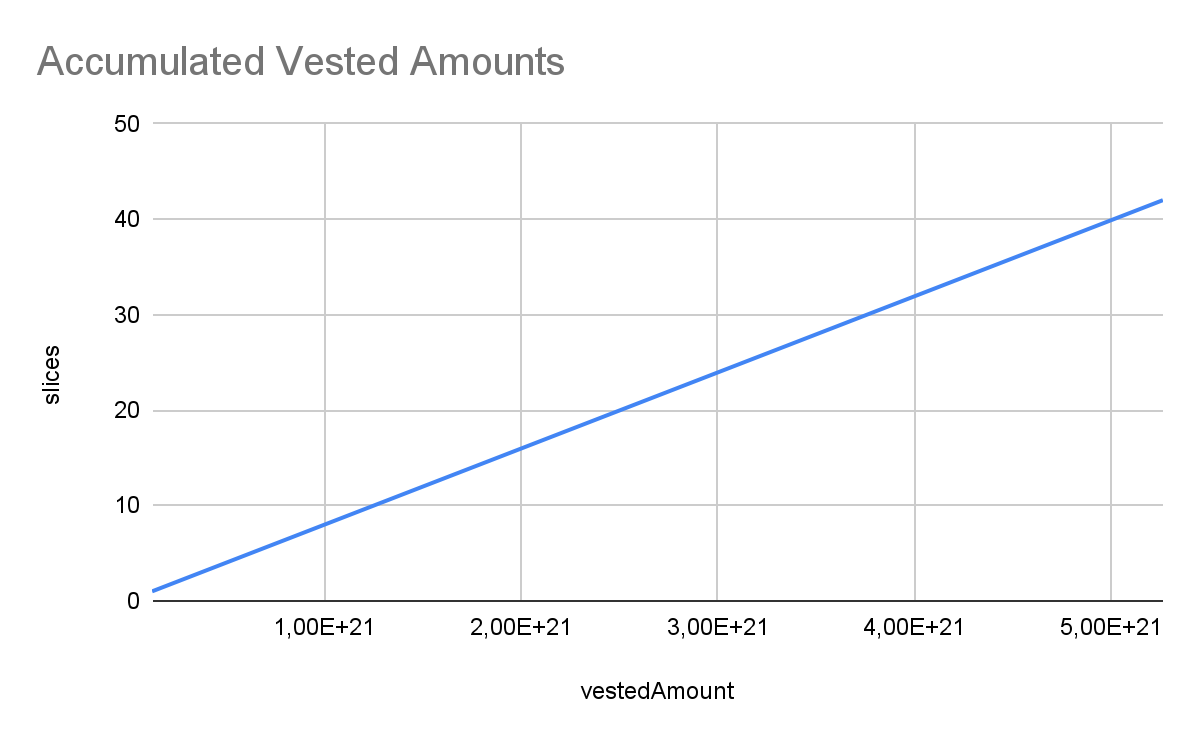
The vesting amount calculation follows a linear distribution from the first slice until the last one (the number of slices is different for each contract). The vesting amount sheet contains the details about the calculations. The X-Axis depicts the accumulative amount. Each user that participated in one or more of the ecosystem’s contracts, receives a proportional amount.

#### DevContract

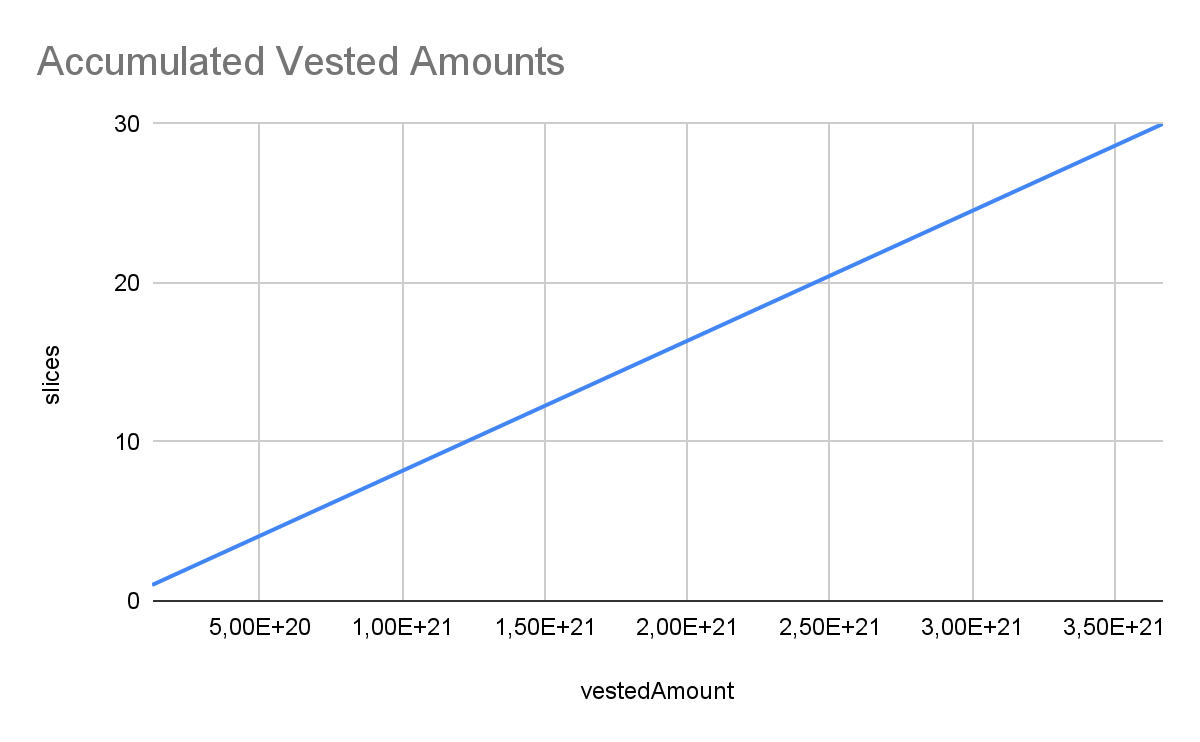


#### 

#### INV\_Seed

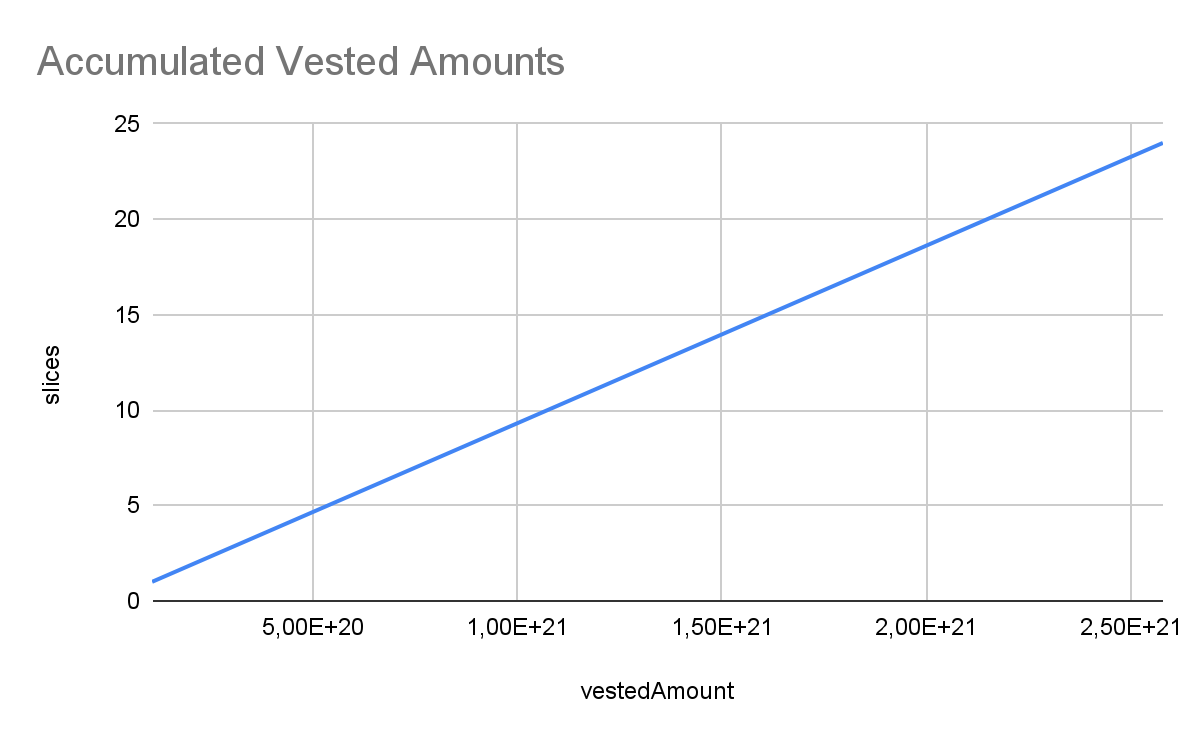


#### INV\_Round1

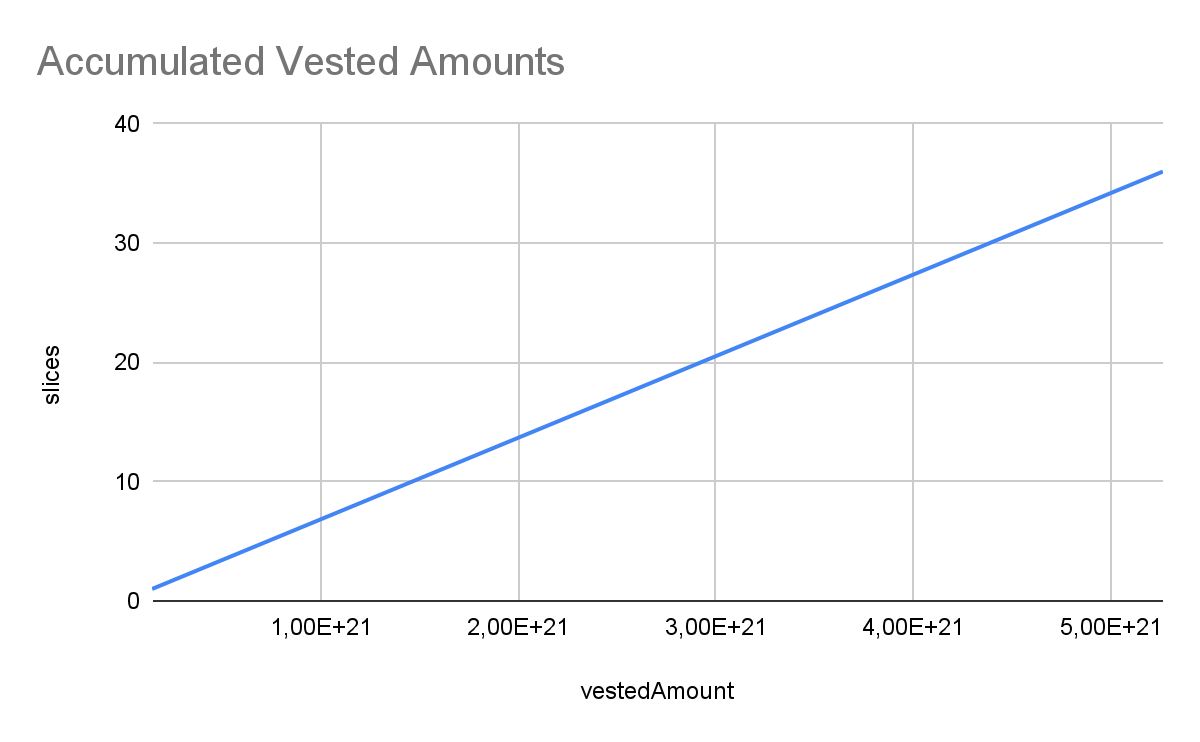


#### 

#### INV\_Round2



#### SeedReplica



### 

### Roles

#### Owner

The owner has the authority to

* Set a new address for the OGBToken and USDToken.
* Add/remove an address from the Operator role.
* Change the time period to release each slice of the vested amount.
* Change the max amount each user can vest.

#### Operator

The operator has the authority to

* Start/stop the ICO process (Initial Coin Offering).

#### User

The user has the authority to

* Deposit an amount in USDT and get OGBToken in return through the vesting process.
* Release the vested amount and get OGB tokens in slices.

## Diagnostics

|  |  |  | ⬤ | Critical | ⬤ | Medium | ⬤ | Minor / Informative |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |

| **Severity** | **Code** | **Description** | **Status** |
| --- | --- | --- | --- |
| ⬤ | CSC | Contract State Configuration | Unresolved |
| ⬤ | SOIL | Stable Operations in Loop | Unresolved |
| ⬤ | GCC | Gas Consuming Calculations | Unresolved |
| ⬤ | MC | Missing Check | Unresolved |
| ⬤ | CR | Calculation Repetition | Unresolved |
| ⬤ | PTAI | Potential Transfer Amount Inconsistency | Unresolved |
| ⬤ | RSML | Redundant SafeMath Library | Unresolved |
| ⬤ | OCTD | Transfers Contract's Tokens | Unresolved |
| ⬤ | UVF | Unused Variables And Functions | Unresolved |
| ⬤ | UF | Unimplemented Function | Unresolved |
| ⬤ | CR | Code Repetition | Unresolved |
| ⬤ | ZD | Zero Division | Unresolved |
| ⬤ | CO | Code Optimization | Unresolved |
| ⬤ | RVA | Redundant Variable Assignment | Unresolved |
| ⬤ | DPI | Decimals Precision Inconsistency | Unresolved |
| ⬤ | TUU | Time Units Usage | Unresolved |
| ⬤ | AAO | Accumulated Amount Overflow | Unresolved |
| ⬤ | L02 | State Variables could be Declared Constant | Unresolved |
| ⬤ | L04 | Conformance to Solidity Naming Conventions | Unresolved |
| ⬤ | L05 | Unused State Variable | Unresolved |
| ⬤ | L07 | Missing Events Arithmetic | Unresolved |
| ⬤ | L09 | Dead Code Elimination | Unresolved |
| ⬤ | L12 | Using Variables before Declaration | Unresolved |
| ⬤ | L13 | Divide before Multiply Operation | Unresolved |
| ⬤ | L14 | Uninitialized Variables in Local Scope | Unresolved |
| ⬤ | L16 | Validate Variable Setters | Unresolved |
| ⬤ | L17 | Usage of Solidity Assembly | Unresolved |
| ⬤ | L19 | Stable Compiler Version | Unresolved |
| ⬤ | L20 | Succeeded Transfer Check | Unresolved |

### 

### CSC - Contract State Configuration

| **Criticality** | Minor / Informative |
| --- | --- |
| **Status** | Unresolved |

#### Description

The contract calculations are heavily dependent on the configuration of the state. For instance, the combination of sliceTime and noOfSlices produced the computedAmount. This amount is added to the lastWithdrawAmount when the user claims the corresponding tokens. If the contract owner increases the sliceTime or noOfSlices then the calculation will yield a lower number than the previous calculations. As a result, the expression computedAmount.sub(lastWithdrawAmount[\_benifierAddress]); will underflow.

#### Recommendation

The team is advised to add sanity checks in the state setter methods in order to prevent the contract from resulting in an odd state.

### 

### SOIL - Stable Operations in Loop

| **Criticality** | Minor / Informative |
| --- | --- |
| **Location** | SeedReplica.sol#L1284 |
| **Status** | Unresolved |

#### Description

The contract contains some checks inside the loop that do not change by the loop. As a result, the expression is executed N times but will always yield the same result.

| for(uint256 x=0; x < \_clientDetailarr.length; x++){  require(!isICOLaunch, "Token launched, cannot deposit");  ... |
| --- |

#### Recommendation

The team is advised to move the stable checks before the loop execution.

### 

### GCC - Gas Consuming Calculations

| **Criticality** | Minor / Informative |
| --- | --- |
| **Status** | Unresolved |

#### Description

The contract provides a different result if the last digit of the number is greater than four. The contract divides, then multiplies, and then subtracts in order to calculate the result. This process increases the gas consumption and complexity in the source code.

| uint256 step1 = OGBToken/10; //5879  uint256 step2 = step1 \* 10;//58790  uint256 step3 = OGBToken - step2;  if(step3 >4){  console.log(step3);  \_result = ((OGBToken.add(10)).mul(tokenGenPercentage))/10\*\*20;  }  else{  \_result = (OGBToken.mul(tokenGenPercentage))/10\*\*20;  } |
| --- |

#### Recommendation

If the modulo 10 is applied to a decimal number, it will yield the last digit number. Thus, the algorithm could be simplified on one expression OGBToken % 10 >4.

### 

### MC - Missing Check

| **Criticality** | Minor / Informative |
| --- | --- |
| **Status** | Unresolved |

#### Description

The contract is processing variables that have not been properly sanitized and checking that they form the proper shape. These variables may produce vulnerability issues.

The variable devFee calculates the fee that will be transferred to the dev wallet. The devFee is divided by 10\*\*18 as a result, if the owner sets a value greater than this, the contract will revert to the expression uint256 \_remaingAmount = \_amount - \_devAmount;

The variable tokenGenPercentage represents a percentage. This variable is dividend by the number 10\*\*20, if the owner sets a value greater than 10\*\*20, then the percentage will be more than 100%.

#### Recommendation

The team is advised to properly check the variables according to the required specifications.

* The devFee should not be greater than 10\*\*18
* The tokenGenPercentage should not be greater than 10\*\*20

### 

### CR - Calculation Repetition

| **Criticality** | Minor / Informative |
| --- | --- |
| **Location** | INV\_Round1.sol#L127  INV\_Round2.sol#L132  SeedReplica.sol#L125  INV\_Seed.sol#L125 |
| **Status** | Unresolved |

#### Description

The contract calculates the ((\_amount.mul(10\*\*18)).div(USDT\_To\_OGB\_Rate)) expression twice. As a result, it increases the gas consumption and creates unessecary complexity in the contract.

| function getAccurateResult(uint256 \_amount) private view returns(uint256 \_result) {    uint256 OGBToken = ((\_amount.mul(10\*\*18)).div(USDT\_To\_OGB\_Rate));  console.log(OGBToken);  uint256 step1 = OGBToken/10; //5879  uint256 step2 = step1 \* 10;//58790  uint256 step3 = OGBToken - step2;  console.log(step3);  if(step3 >4){    \_result = (((\_amount.mul(10\*\*2).add(1)).mul(10\*\*18)).div(USDT\_To\_OGB\_Rate))/10\*\*2;    }  else{  \_result = ((\_amount.mul(10\*\*18)).div(USDT\_To\_OGB\_Rate));  }  } |
| --- |

#### Recommendation

The team is adviced to resuse the initial calculation rather that re-calculating the same expression.

### PTAI - Potential Transfer Amount Inconsistency

| **Criticality** | Minor / Informative |
| --- | --- |
| **Location** | INV\_Round1.sol#L118,119,264,266  INV\_Round2.sol#L121,122,267,269  INV\_Seed.sol#L116,117,262,264  SeedReplica.sol#L1383,1385 |
| **Status** | Unresolved |

#### Description

The transfer() and transferFrom() functions are used to transfer a specified amount of tokens to an address. The fee or tax is an amount that is charged to the sender of an ERC20 token when tokens are transferred to another address. According to the specification, the transferred amount could potentially be less than the expected amount. This may produce inconsistency between the expected and the actual behavior.

The following example depicts the diversion between the expected and actual amount.

| **Tax** | **Amount** | **Expected** | **Actual** |
| --- | --- | --- | --- |
| No Tax | 100 | 100 | 100 |
| 10% Tax | 100 | 100 | 90 |

The contract transfers funds to either a specific wallet address or a user's address upon releasing the vested amount. The USDT or OGB token contracts could potentially charge fees during transactions. As a result, the actual transferred amount may differ than the expected one.

| IERC20(USDTAddress).transferFrom(msg.sender, devWallet, \_devAmount);  IERC20(USDTAddress).transferFrom(msg.sender, OwnerWallet, \_remaingAmount);  ...  if(\_beforeVestingAmount > 0 ){  IERC20(OGBAddress).transfer(\_benifierAddress, \_beforeVestingAmount);  }  IERC20(OGBAddress).transfer(\_benifierAddress, vestedAmount); |
| --- |

#### Recommendation

The team is advised to take into consideration the actual amount that has been transferred instead of the expected.

It is important to note that an ERC20 transfer tax is not a standard feature of the ERC20 specification, and it is not universally implemented by all ERC20 contracts. Therefore, the contract could produce the actual amount by calculating the difference between the transfer call.

Actual Transferred Amount = Balance After Transfer - Balance Before Transfer

### 

### RSML - Redundant SafeMath Library

| **Criticality** | Minor / Informative |
| --- | --- |
| **Location** | DevContract.sol#L5  INV\_Round1.sol#L5  INV\_Round2.sol#L5  INV\_Seed.sol#L5  LaunchContract.sol#L5  SeedReplica.sol#L96 |
| **Status** | Unresolved |

#### Description

SafeMath is a popular Solidity library that provides a set of functions for performing common arithmetic operations in a way that is resistant to integer overflows and underflows.

Starting with Solidity versions that are greater than or equal to 0.8.0, the arithmetic operations revert on underflow and overflow. As a result, the native functionality of the Solidity operations replaces the SafeMath library. Hence, the usage of the SafeMath library adds complexity, overhead and increases unnecessarily the gas consumption.

| import "@openzeppelin/contracts/utils/math/SafeMath.sol";  library SafeMath { ... } |
| --- |

#### Recommendation

The team is advised to remove the SafeMath library. Since the version of the contract is greater than 0.8.0 then the pure Solidity arithmetic operations produce the same result.

If the previous functionality is required, then the contract could exploit the unchecked { ... } statement.

Read more about the breaking change on <https://docs.soliditylang.org/en/v0.8.16/080-breaking-changes.html#solidity-v0-8-0-breaking-changes>.

### OCTD - Transfers Contract's Tokens

| **Criticality** | Minor / Informative |
| --- | --- |
| **Location** | DevContract.sol#L194  INV\_Round1.sol#L333  INV\_Round2.sol#L336  INV\_Seed.sol#L331  LaunchContract.sol#L50  SeedReplica.sol#L1490 |
| **Status** | Unresolved |

#### Description

The contract owner has the authority to claim all the balance of the contract. The owner may take advantage of it by calling the emergencyWithdrawToken function.

| function emergencyWithdrawToken(address token, address destination) public onlyOwner returns (bool sent){  IERC20(token).transfer(destination, IERC20(token).balanceOf(address(this)));  return true;  } |
| --- |

#### Recommendation

The team should carefully manage the private keys of the owner’s account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions. That risk can be prevented by temporarily locking the contract or renouncing ownership.

### 

### UVF - Unused Variables And Functions

| **Criticality** | Minor / Informative |
| --- | --- |
| **Location** | DevContract.sol#L209 |
| **Status** | Unresolved |

#### Description

An unused variable/function is a variable/function that is declared in the contract, but is never used in any of the contract's functions. This can happen if the variable/function was originally intended to be used, but was later removed or never used.

Unused variables/functions can create clutter in the contract and make it more difficult to understand and maintain. They can also increase the size of the contract and the cost of deploying and interacting with it. The contract declares the variable USDTAddress and its value is being assigned by the setUSDTAddress function, but it is not being used anywhere else in the contract.

| address public USDTAddress;  ...  function setUSDTAddress(address \_USDTAddress) public onlyOwner{  USDTAddress = \_USDTAddress;  } |
| --- |

#### Recommendation

To avoid creating unused state variables, it's important to carefully consider the state variables that are needed for the contract's functionality, and to remove any that are no longer needed. This can help improve the clarity and efficiency of the contract.

### 

### UF - Unimplemented Function

| **Criticality** | Minor / Informative |
| --- | --- |
| **Location** | SeedReplica.sol#L1501 |
| **Status** | Unresolved |

#### Description

A function with no code is a function that has been declared in a contract or interface but has no actual code that executes when the function is called. This could happen if a developer forgets to write the code for the function or intentionally leaves the function empty.

While a function with no code won't cause any syntax errors, it will prevent the contract from operating as intended since the function will not execute any actions. When calling a function with no code, the transaction will simply return without doing anything.

| function getRate(uint256 \_amount) public view returns(uint256 OGBAmount){} |
| --- |

#### Recommendation

The team is advised to ensure that the function has the necessary code to execute the desired actions when called. If the function is intentionally left empty, it is important to document why it is empty to avoid confusion for other people who may work on the contract or interact with it in the future.

### 

### CR - Code Repetition

| **Criticality** | Minor / Informative |
| --- | --- |
| **Location** | DevContract.sol#L59, |
| **Status** | Unresolved |

#### Description

The contract contains repetitive code segments. There are potential issues that can arise when using code segments in Solidity. Some of them can lead to issues like gas efficiency, complexity, readability, security, and maintainability of the source code. It is generally a good idea to try to minimize code repetition where possible.

The contract can assign or modify a new value to the OGBAddress variable in two different functions. As a result, the same functionality is duplicated.

| function setTokenAddress(address \_OGBAddress) public onlyOwner{  OGBAddress = \_OGBAddress;  }  ...  function setOGBTAddress(address \_OGBAddress) public onlyOwner{  OGBAddress = \_OGBAddress;  } |
| --- |

#### Recommendation

The team is advised to avoid repeating the same code in multiple places, which can make the contract easier to read and maintain. The authors could try to reuse code wherever possible, as this can help to reduce the complexity and size of the contract. For instance, the contract could reuse the common code segments in an internal function in order to avoid repeating the same code in multiple places.

### 

### ZD - Zero Division

| **Criticality** | Critical |
| --- | --- |
| **Location** | DevContract.sol#L129,174  INV\_Seed.sol#L127,158,291,318  INV\_Round1.sol#L129,160,293,320  INV\_Round2.sol#L132,163,296,323  SeedReplica.sol#L1305,1336,1412,1439 |
| **Status** | Unresolved |

#### Description

The contract is using variables that may be set to zero as denominators. This can lead to unpredictable and potentially harmful results, such as a transaction revert. Overall, the following variables can be set to zero or there can be a scenario where they haven't been initialized with a value:

* sliceTime
* USDT\_To\_OGB\_Rate
* noOfSlices

Regarding the sliceTime variable, it will be zero in the following scenario:

1. initialize() function isn't called after contracts deployment, so sliceTime value remains zero.
2. deposit() function is called.
3. release() is called, lastly, resulting in a division by zero.

| uint256 NumberOfSlicesPassed = timePassed.div(sliceTime);  ...  uint256 OGBToken = ((\_amount.mul(10\*\*18)).div(USDT\_To\_OGB\_Rate));  ...  \_perSliceTokens = (\_totalOGBTokens).div(noOfSlices);  uint256 amountPerSlice = (\_amountToHold).div(noOfSlices);  uint256 perSliceTime = (endTime - startTime).div(noOfSlices); |
| --- |

#### 

#### Recommendation

It is important to handle division by zero appropriately in the code to avoid unintended behavior and to ensure the reliability and safety of the contract. The contract should ensure that the divisor is always non-zero before performing a division operation. It should prevent the variables to be set to zero or should not allow executing of the corresponding statements.

### 

### CO - Code Optimization

| **Criticality** | Minor / Informative |
| --- | --- |
| **Location** | INV\_Seed.sol#L268  INV\_Round1.sol#L270  INV\_Round2.sol#L273  SeedReplica.sol#L1389 |
| **Status** | Unresolved |

#### Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The contract performs the same operation at the if block and the else-if block while checking the same condition. As a result, the code is being repeated and makes it more difficult to undestand.

| else if(\_beforeVestingAmount > 0){  IERC20(OGBAddress).transfer(\_benifierAddress, \_beforeVestingAmount);  beforeVestingTokens[\_benifierAddress] = 0;  } |
| --- |

#### Recommendation

The team is advised to take into consideration these segments and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

### 

### RVA - Redundant Variable Assignment

| **Criticality** | Minor / Informative |
| --- | --- |
| **Location** | SeedReplica.sol#L1285 |
| **Status** | Unresolved |

#### Description

The contract assigns the \_clientDetail array to a new variable and stored in memory. This new variable is not being manipulated in any way, instead it is only used for reading. This can become a problem in situations where memory usage is a concern, such as when working with large data sets since gas fees are based on the amount of memory used.

| ClientDetail[] memory \_clientDetailarr = \_clientDetail; |
| --- |

#### Recommendation

The team is advised to remove the array's reassignment completely, since it is redundant and may produce additional gas fees. In general, it is recommended to assign a value to a single variable and then use that variable in subsequent operations.

### 

### DPI - Decimals Precision Inconsistency

| **Criticality** | Medium |
| --- | --- |
| **Location** | INV\_Seed.sol#L261  DevContract.sol#L104  INV\_Round1.sol#L263  INV\_Round2.sol#L266  SeedReplica.sol#L1382 |
| **Status** | Unresolved |

#### Description

The decimals field of a contract's ERC20 token can be used to specify the number of decimal places that the token uses. For example, if decimals is set to 8, it means that the smallest unit of the token is 0.00000001, and if decimals are set to 18, it means that the smallest unit of the token is 0.000000000000000001.

However, there is an inconsistency in the way that the decimals field is handled in some ERC20 contracts. The ERC20 specification does not specify how the decimals field should be implemented, and as a result, some contracts use different precision numbers.

This inconsistency can cause problems when interacting with these contracts, as it is not always clear how the decimals field should be interpreted. For example, if a contract expects the decimals field to be 18 digits, but the contract being interacted with uses 8 digits, the result of the interaction may not be what was expected.

The contract engages with two distinct contracts, transferring funds to them, and each contract may anticipate a different number of decimal places compared to the original contract. As a result, this can lead to inconsistencies in the transferred amounts.

| if(\_beforeVestingAmount > 0 ){  IERC20(OGBAddress).transfer(\_benifierAddress, \_beforeVestingAmount);  }  IERC20(OGBAddress).transfer(\_benifierAddress, vestedAmount); |
| --- |

#### 

#### Recommendation

To avoid these issues, it is important to carefully review the implementation of the decimals field of the underlying tokens. The team is advised to normalize each decimal to one single source of truth. A recommended way is to scale all the decimals to the greatest token's decimal. Hence, the contract will not lose precision in the calculations.

The following example depicts 3 tokens with different decimals precision.

| **ERC20** | **Decimals** |
| --- | --- |
| Token 1 | 6 |
| Token 2 | 9 |
| Token 3 | 18 |

All the decimals could be normalized to 18 since it represents the ERC20 token with the greatest digits.

### 

### TUU - Time Units Usage

| **Criticality** | Minor / Informative |
| --- | --- |
| **Location** | DevContract.sol#L52,53,55  INV\_Seed.sol#L66,69  INV\_Round1.sol#L68,71  INV\_Round2.sol#L71,74  SeedReplica.sol#L1244,1247 |
| **Status** | Unresolved |

#### Description

The contract is using arbitrary numbers to form time-related values. As a result, it decreases the readability of the codebase and prevents the compiler to optimize the source code.

| timePeriod = 12600;  sliceTime = 300;  cliff = block.timestamp + 360;  ... |
| --- |

#### Recommendation

It is a good practice to use the time units reserved keywords like seconds, minutes, hours, days, weeks and years to process time-related calculations.

It's important to note that these time units are simply a shorthand notation for representing time in seconds, and do not have any effect on the actual passage of time or the execution of the contract. The time units are simply a convenience for expressing time in a more human-readable form.

### 

### AAO - Accumulated Amount Overflow

| **Criticality** | Minor / Informative |
| --- | --- |
| **Location** | INV\_Seed.sol#L114  INV\_Round1.sol#L116  INV\_Round2.sol#L119 |
| **Status** | Unresolved |

#### Description

The contract is using variables to accumulate values. The contract could lead to an overflow when the total value of a variable exceeds the maximum value that can be stored in that variable's data type. This can happen when an accumulated value is updated repeatedly over time, and the value grows beyond the maximum value that can be represented by the data type.

The value of bookedTokens variable is increased each time the deposit() function is called. The value that is added to the bookedTokens is dependent on the deposited amount and, most importantly, the USDT\_To\_OGB\_Rate variable. The lower the USDT\_To\_OGB\_Rate value is, the higher the amount that is added to the bookedTokens will be. As a result, the accumulated amount may potentially lead to an overflow.

| bookedTokens += accurateOGBToken; |
| --- |

#### Recommendation

The team is advised to carefully investigate the usage of the variables that accumulate value. A suggestion is to add checks to the code to ensure that the value of a variable does not exceed the maximum value that can be stored in its data type.

### 

### L02 - State Variables could be Declared Constant

| **Criticality** | Minor / Informative |
| --- | --- |
| **Location** | LaunchContract.sol#L23  DevContract.sol#L26,27 |
| **Status** | Unresolved |

#### Description

State variables can be declared as constant using the constant keyword. This means that the value of the state variable cannot be changed after it has been set. Additionally, the constant variables decrease gas consumption of the corresponding transaction.

| address public round2Address  uint256 public perSlicePercentage  uint256 public devFee |
| --- |

#### Recommendation

Constant state variables can be useful when the contract wants to ensure that the value of a state variable cannot be changed by any function in the contract. This can be useful for storing values that are important to the contract's behavior, such as the contract's address or the maximum number of times a certain function can be called. The team is advised to add the constant keyword to state variables that never change.

### 

### L04 - Conformance to Solidity Naming Conventions

| **Criticality** | Minor / Informative |
| --- | --- |
| **Location** | SeedReplica.sol#L649,652,667,692,696,757,889,892,1079,1097,1100,1103,1181,1189,1190,1200,1205,1207,1210,1225,1231,1253,1257,1261,1267,1273,1279,1303,1320,1333,1339,1343,1351,1356,1360,1371,1418,1448  LaunchContract.sol#L12,33,39  INV\_Seed.sol#L10,14,15,24,25,27,30,45,51,72,76,80,86,92,98,103,125,142,155,161,165,173,236,240,251,297,327,342  INV\_Round2.sol#L11,15,16,25,26,28,31,50,56,77,81,85,91,97,103,108,130,147,160,166,170,178,241,245,256,302,332  INV\_Round1.sol#L11,15,16,25,26,28,32,47,53,74,78,82,88,94,100,105,127,144,157,163,167,175,238,242,253,299,329  DevContract.sol#L11,21,22,39,59,66,155,190,205,209 |
| **Status** | Unresolved |

#### Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX\_VALUE, ERROR\_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

| function \_\_Context\_init() internal onlyInitializing {  }  function \_\_Context\_init\_unchained() internal onlyInitializing {  }  uint256[50] private \_\_gap  ...  function \_\_Ownable\_init\_unchained() internal onlyInitializing {  \_transferOwnership(\_msgSender());  }  uint256[49] private \_\_gap  function \_\_ERC1967Upgrade\_init() internal onlyInitializing {  }  ... |
| --- |

#### Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.

Find more information on the Solidity documentation [https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention](https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-conventions).

### 

### L05 - Unused State Variable

| **Criticality** | Minor / Informative |
| --- | --- |
| **Location** | SeedReplica.sol#L1181 |
| **Status** | Unresolved |

#### Description

An unused state variable is a state variable that is declared in the contract, but is never used in any of the contract's functions. This can happen if the state variable was originally intended to be used, but was later removed or never used.

Unused state variables can create clutter in the contract and make it more difficult to understand and maintain. They can also increase the size of the contract and the cost of deploying and interacting with it.

| uint256[50] private \_\_gap |
| --- |

#### Recommendation

To avoid creating unused state variables, it's important to carefully consider the state variables that are needed for the contract's functionality and to remove any that are no longer needed. This can help improve the clarity and efficiency of the contract.

### 

### L07 - Missing Events Arithmetic

| **Criticality** | Minor / Informative |
| --- | --- |
| **Location** | SeedReplica.sol#L1254,1280,1357  INV\_Seed.sol#L73,77,99,237  INV\_Round2.sol#L78,82,104,242  INV\_Round1.sol#L75,79,101,239 |
| **Status** | Unresolved |

#### Description

Events are a way to record and log information about changes or actions that occur within a contract. They are often used to notify external parties or clients about events that have occurred within the contract, such as the transfer of tokens or the completion of a task.

It's important to carefully design and implement the events in a contract, and to ensure that all required events are included. It's also a good idea to test the contract to ensure that all events are being properly triggered and logged.

| sliceTime = \_timeSeconds  USDT\_To\_OGB\_Rate = \_rate  noOfSlices = \_slices  usdtLimit = \_amount |
| --- |

#### Recommendation

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues with its arithmetic.

### 

### L09 - Dead Code Elimination

| **Criticality** | Minor / Informative |
| --- | --- |
| **Location** | SeedReplica.sol#L365,390,400,419,433,452,462,649,652,860,870,889,892,990,997,1007,1026,1033,1048,1100 |
| **Status** | Unresolved |

#### Description

In Solidity, dead code is code that is written in the contract, but is never executed or reached during normal contract execution. Dead code can occur for a variety of reasons, such as:

* Conditional statements that are always false.
* Functions that are never called.
* Unreachable code (e.g., code that follows a return statement).

Dead code can make a contract more difficult to understand and maintain, and can also increase the size of the contract and the cost of deploying and interacting with it.

| function sendValue(address payable recipient, uint256 amount) internal {  require(address(this).balance >= amount, "Address: insufficient balance");  (bool success, ) = recipient.call{value: amount}("");  require(success, "Address: unable to send value, recipient may have reverted");  }  function functionCall(address target, bytes memory data) internal returns (bytes memory) {  return functionCall(target, data, "Address: low-level call failed");  }  ... |
| --- |

#### 

#### Recommendation

To avoid creating dead code, it's important to carefully consider the logic and flow of the contract and to remove any code that is not needed or that is never executed. This can help improve the clarity and efficiency of the contract.

### 

### L12 - Using Variables before Declaration

| **Criticality** | Minor / Informative |
| --- | --- |
| **Location** | SeedReplica.sol#L966 |
| **Status** | Unresolved |

#### Description

The contract is using a variable before the declaration. This is usually happening either if it has not been declared yet or if the variable has been declared in a different scope. It is not a good practice to use a local variable before it has been declared.

| bytes32 slot |
| --- |

#### Recommendation

By declaring local variables before using them, contract ensures that it operates correctly. It's important to be aware of this rule when working with local variables, as using a variable before it has been declared can lead to unexpected behavior and can be difficult to debug.

### 

### L13 - Divide before Multiply Operation

| **Criticality** | Minor / Informative |
| --- | --- |
| **Location** | SeedReplica.sol#L1307,1308,1322,1323,1410,1412,1414,1439,1441  INV\_Seed.sol#L129,130,144,145,289,291,293,318,320  INV\_Round2.sol#L134,135,149,150,294,296,298,323,325  INV\_Round1.sol#L131,132,146,147,291,293,295,320,322  DevContract.sol#L129,139,141 |
| **Status** | Unresolved |

#### Description

It is important to be aware of the order of operations when performing arithmetic calculations. This is especially important when working with large numbers, as the order of operations can affect the final result of the calculation. Performing divisions before multiplications may cause loss of prediction.

| uint256 step1 = OGBToken/10  uint256 step2 = step1 \* 10  uint256 NumberOfSlicesPassed = timePassed.div(sliceTime)  uint256 amountPerSlice = (\_amountToHold).div(noOfSlices)  return (amountPerSlice.mul(NumberOfSlicesPassed), NumberOfSlicesPassed)  uint256 NumberOfSlicesPassed = timePassed.div(sliceTime)  uint256 amountPerSlice = (\_amountToHold).div(noOfSlices)  \_amount = amountPerSlice.mul(NumberOfSlicesPassed) |
| --- |

#### Recommendation

To avoid this issue, it is recommended to carefully consider the order of operations when performing arithmetic calculations in Solidity. It's generally a good idea to use parentheses to specify the order of operations. The basic rule is that the multiplications should be prior to the divisions.

### 

### L14 - Uninitialized Variables in Local Scope

| **Criticality** | Minor / Informative |
| --- | --- |
| **Location** | SeedReplica.sol#L966 |
| **Status** | Unresolved |

#### Description

Using an uninitialized local variable can lead to unpredictable behavior and potentially cause errors in the contract. It's important to always initialize local variables with appropriate values before using them.

| bytes32 slot |
| --- |

#### Recommendation

By initializing local variables before using them, the contract ensures that the functions behave as expected and avoid potential issues.

### 

### L16 - Validate Variable Setters

| **Criticality** | Minor / Informative |
| --- | --- |
| **Location** | SeedReplica.sol#L1234,1235,1262,1497  LaunchContract.sol#L34,35,57  INV\_Seed.sol#L54,55,56,81,338,343  INV\_Round2.sol#L59,60,61,86,343  INV\_Round1.sol#L56,57,58,83,340  DevContract.sol#L60,67,201,206,210 |
| **Status** | Unresolved |

#### Description

The contract performs operations on variables that have been configured on user-supplied input. These variables are missing of proper check for the case where a value is zero. This can lead to problems when the contract is executed, as certain actions may not be properly handled when the value is zero.

| OGBAddress = \_OGBAddress  launchContract = \_launchContract  OwnerWallet = \_ownerAddress  payable(destination).transfer(address(this).balance)  seedAddress = \_seedContract  round1Address = \_round1  USDTAddress = \_USDTAddress |
| --- |

#### Recommendation

By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

### 

### L17 - Usage of Solidity Assembly

| **Criticality** | Minor / Informative |
| --- | --- |
| **Location** | SeedReplica.sol#L491,842,852,862,872 |
| **Status** | Unresolved |

#### Description

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Some common types of errors that can occur when using assembly in Solidity include Syntax, Type, Out-of-bounds, Stack, and Revert.

| assembly {  let returndata\_size := mload(returndata)  revert(add(32, returndata), returndata\_size)  }  assembly {  r.slot := slot  } |
| --- |

#### Recommendation

It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

### 

### L19 - Stable Compiler Version

| **Criticality** | Minor / Informative |
| --- | --- |
| **Location** | SeedReplica.sol#L2  LaunchContract.sol#L2  INV\_Seed.sol#L2  INV\_Round2.sol#L2  INV\_Round1.sol#L2  DevContract.sol#L2 |
| **Status** | Unresolved |

#### Description

The ^ symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

| pragma solidity ^0.8.7; |
| --- |

#### Recommendation

The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

### 

### L20 - Succeeded Transfer Check

| **Criticality** | Minor / Informative |
| --- | --- |
| **Location** | SeedReplica.sol#L1383,1385,1390,1491  LaunchContract.sol#L51  INV\_Seed.sol#L116,117,262,264,269,332  INV\_Round2.sol#L121,122,267,269,274,337  INV\_Round1.sol#L118,119,264,266,271,334  DevContract.sol#L70,104,195 |
| **Status** | Unresolved |

#### Description

According to the ERC20 specification, the transfer methods should be checked if the result is successful. Otherwise, the contract may wrongly assume that the transfer has been established.

| IERC20(OGBAddress).transfer(\_benifierAddress, \_beforeVestingAmount)  IERC20(OGBAddress).transfer(\_benifierAddress, vestedAmount)  IERC20(token).transfer(destination, IERC20(token).balanceOf(address(this)))  IERC20(USDTAddress).transferFrom(msg.sender, devWallet, \_devAmount)  IERC20(USDTAddress).transferFrom(msg.sender, OwnerWallet, \_remaingAmount)  IERC20(OGBAddress).transferFrom(OGBAddress, address(this), \_amount)  IERC20(OGBAddress).transfer(msg.sender, vestedAmount) |
| --- |

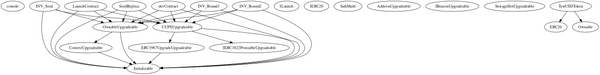
#### Recommendation

The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the [Openzeppelin library](https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/utils/SafeERC20.sol).

## Functions Analysis

| **Contract** | **Type** | **Bases** |  |  |
| --- | --- | --- | --- | --- |
|  | **Function Name** | **Visibility** | **Mutability** | **Modifiers** |
| **devContract** | Implementation | Initializable, OwnableUpgradeable, UUPSUpgradeable |  |  |
|  |  | Public️ | ✓ | -️ |
|  | initialize | Public️ | ✓ | initializer |
|  | setTokenAddress | Public️ | ✓ | onlyOwner |
|  | \_authorizeUpgrade | Internal | ✓ | onlyOwner |
|  | deposit | Public️ | ✓ | -️ |
|  | getVestedAmount | Public️ |  | -️ |
|  | release | Public️ | ✓ | onlyOwner |
|  | \_computeReleasableAmount | Private |  |  |
|  | V\_GetRemaingTime | Public️ |  | -️ |
|  | getTokenBalance | Public️ |  | -️ |
|  | emergencyWithdrawToken | Public️ | ✓ | onlyOwner |
|  | emergencyWithdrawCurrency | Public️ | ✓ | onlyOwner |
|  | setOGBTAddress | Public️ | ✓ | onlyOwner |
|  | setUSDTAddress | Public️ | ✓ | onlyOwner |
|  |  | External️ | Payable | -️ |
|  |  | External️ | Payable | -️ |
|  |  |  |  |  |
| **INV\_Round1** | Implementation | Initializable, OwnableUpgradeable, UUPSUpgradeable |  |  |
|  |  | Public️ | ✓ | -️ |
|  | initialize | Public️ | ✓ | initializer |
|  | setSliceTime | Public️ | ✓ | onlyOwner |
|  | setUSDTLimit | Public️ | ✓ | onlyOwner |
|  | setOwnerWallet | Public️ | ✓ | onlyOwner |
|  | \_authorizeUpgrade | Internal | ✓ | onlyOwner |
|  | StartICO | Public️ | ✓ | -️ |
|  | StopICO | Public️ | ✓ | -️ |
|  | setUSDT\_OBG\_Rate | Public️ | ✓ | onlyOwner |
|  | deposit | Public️ | ✓ | -️ |
|  | getAccurateResult | Private |  |  |
|  | getAccurateResult2 | Private |  |  |
|  | getUserAmounts | Public️ |  | -️ |
|  | setTimePeriod | Public️ | ✓ | onlyOwner |
|  | Launch | Public️ | ✓ | -️ |
|  | setOperator | Public️ | ✓ | onlyOwner |
|  | setNumberOfSlices | Public️ | ✓ | onlyOwner |
|  | getVestedAmount | Public️ |  | -️ |
|  | release | Public️ | ✓ | -️ |
|  | \_computeReleasableAmount | Private |  |  |
|  | V\_GetRemaingTime | Public️ |  | -️ |
|  | getTokenBalance | Public️ |  | -️ |
|  | emergencyWithdrawToken | Public️ | ✓ | onlyOwner |
|  | emergencyWithdrawCurrency | Public️ | ✓ | onlyOwner |
|  |  | External️ | Payable | -️ |
|  |  | External️ | Payable | -️ |
|  |  |  |  |  |
| **INV\_Round2** | Implementation | Initializable, OwnableUpgradeable, UUPSUpgradeable |  |  |
|  |  | Public️ | ✓ | -️ |
|  | initialize | Public️ | ✓ | initializer |
|  | setSliceTime | Public️ | ✓ | onlyOwner |
|  | setUSDTLimit | Public️ | ✓ | onlyOwner |
|  | setOwnerWallet | Public️ | ✓ | onlyOwner |
|  | \_authorizeUpgrade | Internal | ✓ | onlyOwner |
|  | StartICO | Public️ | ✓ | -️ |
|  | StopICO | Public️ | ✓ | -️ |
|  | setUSDT\_OBG\_Rate | Public️ | ✓ | onlyOwner |
|  | deposit | Public️ | ✓ | -️ |
|  | getAccurateResult | Private |  |  |
|  | getAccurateResult2 | Private |  |  |
|  | getUserAmounts | Public️ |  | -️ |
|  | setTimePeriod | Public️ | ✓ | onlyOwner |
|  | Launch | Public️ | ✓ | -️ |
|  | setOperator | Public️ | ✓ | onlyOwner |
|  | setNumberOfSlices | Public️ | ✓ | onlyOwner |
|  | getVestedAmount | Public️ |  | -️ |
|  | release | Public️ | ✓ | -️ |
|  | \_computeReleasableAmount | Private |  |  |
|  | V\_GetRemaingTime | Public️ |  | -️ |
|  | getTokenBalance | Public️ |  | -️ |
|  | emergencyWithdrawToken | Public️ | ✓ | onlyOwner |
|  | emergencyWithdrawCurrency | Public️ | ✓ | onlyOwner |
|  |  | External️ | Payable | -️ |
|  |  | External️ | Payable | -️ |
|  |  |  |  |  |
| **INV\_Seed** | Implementation | Initializable, OwnableUpgradeable, UUPSUpgradeable |  |  |
|  |  | Public️ | ✓ | -️ |
|  | initialize | Public️ | ✓ | initializer |
|  | setSliceTime | Public️ | ✓ | onlyOwner |
|  | setUSDTLimit | Public️ | ✓ | onlyOwner |
|  | setOwnerWallet | Public️ | ✓ | onlyOwner |
|  | \_authorizeUpgrade | Internal | ✓ | onlyOwner |
|  | StartICO | Public️ | ✓ | -️ |
|  | StopICO | Public️ | ✓ | -️ |
|  | setUSDT\_OBG\_Rate | Public️ | ✓ | onlyOwner |
|  | deposit | Public️ | ✓ | -️ |
|  | getAccurateResult | Private |  |  |
|  | getAccurateResult2 | Private |  |  |
|  | getUserAmounts | Public️ |  | -️ |
|  | setTimePeriod | Public️ | ✓ | onlyOwner |
|  | Launch | Public️ | ✓ | -️ |
|  | setOperator | Public️ | ✓ | onlyOwner |
|  | setNumberOfSlices | Public️ | ✓ | onlyOwner |
|  | getVestedAmount | Public️ |  | -️ |
|  | release | Public️ | ✓ | -️ |
|  | \_computeReleasableAmount | Private |  |  |
|  | V\_GetRemaingTime | Public️ |  | -️ |
|  | getTokenBalance | Public️ |  | -️ |
|  | emergencyWithdrawToken | Public️ | ✓ | onlyOwner |
|  | emergencyWithdrawCurrency | Public️ | ✓ | onlyOwner |
|  | setOGBToken | Public️ | ✓ | onlyOwner |
|  |  | External️ | Payable | -️ |
|  |  | External️ | Payable | -️ |
|  |  |  |  |  |
| **ILaunch** | Interface |  |  |  |
|  | Launch | External️ | ✓ | -️ |
|  |  |  |  |  |
| **LaunchContract** | Implementation | Initializable, OwnableUpgradeable, UUPSUpgradeable |  |  |
|  |  | Public️ | ✓ | -️ |
|  | initialize | Public️ | ✓ | initializer |
|  | \_authorizeUpgrade | Internal | ✓ | onlyOwner |
|  | setContractAddresses | Public️ | ✓ | onlyOwner |
|  | LaunchToken | Public️ | ✓ | onlyOwner |
|  | chkLaunch | Public️ |  | -️ |
|  | emergencyWithdrawToken | Public️ | ✓ | onlyOwner |
|  | emergencyWithdrawCurrency | Public️ | ✓ | onlyOwner |
|  |  | External️ | Payable | -️ |
|  |  | External️ | Payable | -️ |
|  |  |  |  |  |
| **IERC20** | Interface |  |  |  |
|  | totalSupply | External️ |  | -️ |
|  | balanceOf | External️ |  | -️ |
|  | transfer | External️ | ✓ | -️ |
|  | allowance | External️ |  | -️ |
|  | approve | External️ | ✓ | -️ |
|  | transferFrom | External️ | ✓ | -️ |
|  |  |  |  |  |
| **SafeMath** | Library |  |  |  |
|  | tryAdd | Internal |  |  |
|  | trySub | Internal |  |  |
|  | tryMul | Internal |  |  |
|  | tryDiv | Internal |  |  |
|  | tryMod | Internal |  |  |
|  | add | Internal |  |  |
|  | sub | Internal |  |  |
|  | mul | Internal |  |  |
|  | div | Internal |  |  |
|  | mod | Internal |  |  |
|  | sub | Internal |  |  |
|  | div | Internal |  |  |
|  | mod | Internal |  |  |
|  |  |  |  |  |
| **AddressUpgradeable** | Library |  |  |  |
|  | isContract | Internal |  |  |
|  | sendValue | Internal | ✓ |  |
|  | functionCall | Internal | ✓ |  |
|  | functionCall | Internal | ✓ |  |
|  | functionCallWithValue | Internal | ✓ |  |
|  | functionCallWithValue | Internal | ✓ |  |
|  | functionStaticCall | Internal |  |  |
|  | functionStaticCall | Internal |  |  |
|  | verifyCallResult | Internal |  |  |
|  |  |  |  |  |
| **Initializable** | Implementation |  |  |  |
|  | \_disableInitializers | Internal | ✓ |  |
|  |  |  |  |  |
| **ContextUpgradeable** | Implementation | Initializable |  |  |
|  | \_\_Context\_init | Internal | ✓ | onlyInitializing |
|  | \_\_Context\_init\_unchained | Internal | ✓ | onlyInitializing |
|  | \_msgSender | Internal |  |  |
|  | \_msgData | Internal |  |  |
|  |  |  |  |  |
| **OwnableUpgradeable** | Implementation | Initializable, ContextUpgradeable |  |  |
|  | \_\_Ownable\_init | Internal | ✓ | onlyInitializing |
|  | \_\_Ownable\_init\_unchained | Internal | ✓ | onlyInitializing |
|  | owner | Public️ |  | -️ |
|  | \_checkOwner | Internal |  |  |
|  | renounceOwnership | Public️ | ✓ | onlyOwner |
|  | transferOwnership | Public️ | ✓ | onlyOwner |
|  | \_transferOwnership | Internal | ✓ |  |
|  |  |  |  |  |
| **IERC1822ProxiableUpgradeable** | Interface |  |  |  |
|  | proxiableUUID | External️ |  | -️ |
|  |  |  |  |  |
| **IBeaconUpgradeable** | Interface |  |  |  |
|  | implementation | External️ |  | -️ |
|  |  |  |  |  |
| **StorageSlotUpgradeable** | Library |  |  |  |
|  | getAddressSlot | Internal |  |  |
|  | getBooleanSlot | Internal |  |  |
|  | getBytes32Slot | Internal |  |  |
|  | getUint256Slot | Internal |  |  |
|  |  |  |  |  |
| **ERC1967UpgradeUpgradeable** | Implementation | Initializable |  |  |
|  | \_\_ERC1967Upgrade\_init | Internal | ✓ | onlyInitializing |
|  | \_\_ERC1967Upgrade\_init\_unchained | Internal | ✓ | onlyInitializing |
|  | \_getImplementation | Internal |  |  |
|  | \_setImplementation | Private | ✓ |  |
|  | \_upgradeTo | Internal | ✓ |  |
|  | \_upgradeToAndCall | Internal | ✓ |  |
|  | \_upgradeToAndCallUUPS | Internal | ✓ |  |
|  | \_getAdmin | Internal |  |  |
|  | \_setAdmin | Private | ✓ |  |
|  | \_changeAdmin | Internal | ✓ |  |
|  | \_getBeacon | Internal |  |  |
|  | \_setBeacon | Private | ✓ |  |
|  | \_upgradeBeaconToAndCall | Internal | ✓ |  |
|  | \_functionDelegateCall | Private | ✓ |  |
|  |  |  |  |  |
| **UUPSUpgradeable** | Implementation | Initializable, IERC1822ProxiableUpgradeable, ERC1967UpgradeUpgradeable |  |  |
|  | \_\_UUPSUpgradeable\_init | Internal | ✓ | onlyInitializing |
|  | \_\_UUPSUpgradeable\_init\_unchained | Internal | ✓ | onlyInitializing |
|  | proxiableUUID | External️ |  | notDelegated |
|  | upgradeTo | External️ | ✓ | onlyProxy |
|  | upgradeToAndCall | External️ | Payable | onlyProxy |
|  | \_authorizeUpgrade | Internal | ✓ |  |
|  |  |  |  |  |
| **SeedReplica** | Implementation | Initializable, OwnableUpgradeable, UUPSUpgradeable |  |  |
|  |  | Public️ | ✓ | -️ |
|  | initialize | Public️ | ✓ | initializer |
|  | setSliceTime | Public️ | ✓ | onlyOwner |
|  | setUSDTLimit | Public️ | ✓ | onlyOwner |
|  | setOwnerWallet | Public️ | ✓ | onlyOwner |
|  | \_authorizeUpgrade | Internal | ✓ | onlyOwner |
|  | StartICO | Public️ | ✓ | -️ |
|  | StopICO | Public️ | ✓ | -️ |
|  | setUSDT\_OBG\_Rate | Public️ | ✓ | onlyOwner |
|  | depositPrePaid | Public️ | ✓ | onlyOwner |
|  | getAccurateRate | Private |  |  |
|  | getBeforeVestingToken | Private |  |  |
|  | getUserAmounts | Public️ |  | -️ |
|  | setTimePeriod | Public️ | ✓ | onlyOwner |
|  | Launch | Public️ | ✓ | -️ |
|  | setOperator | Public️ | ✓ | onlyOwner |
|  | setNumberOfSlices | Public️ | ✓ | onlyOwner |
|  | getVestedAmount | Public️ |  | -️ |
|  | release | Public️ | ✓ | -️ |
|  | \_computeReleasableAmount | Private |  |  |
|  | V\_GetRemaingTime | Public️ |  | -️ |
|  | getTokenBalance | Public️ |  | -️ |
|  | getClientRecord | Private | ✓ |  |
|  | showClientData | Public️ |  | -️ |
|  | emergencyWithdrawToken | Public️ | ✓ | onlyOwner |
|  | emergencyWithdrawCurrency | Public️ | ✓ | onlyOwner |
|  | getRate | Public️ |  | -️ |
|  |  | External️ | Payable | -️ |
|  |  | External️ | Payable | -️ |
|  |  |  |  |  |
| **TestUSDToken** | Implementation | ERC20, Ownable |  |  |
|  |  | Public️ | ✓ | ERC20 |

## Inheritance Graph



## 

## Flow Graph



## Summary

OpenGames contract implements a financial, staking, and utility mechanism. This audit investigates security issues, business logic concerns, and potential improvements.

## Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope’s prior written consent. This report is not nor should be considered an “endorsement” or “disapproval” of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope’s position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

## About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors’ funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.



The Cyberscope team

<https://www.cyberscope.io>