Cyberscope

Audit Report

Vestor

October 2022

Github https://github.com/vestor-co/vestor-contracts

Commit 9814dd933047ace2f99bf56e9a239bfe09ff785e

Audited by © cyberscope

Table of Contents

Contract Review

Contract Name	vestor
Compiler Version	v0.8.11+commit.d7f03943
Github	https://github.com/vestor-co/vestor-contracts
Commit	9814dd933047ace2f99bf56e9a239bfe09ff785e

Audit Updates

Initial Audit	12th October 2022
Corrected	

Source Files

Filename	SHA256
@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol	cd823c76cbf5f5b6ef1bda565d58be66c843c37707cd93eb8fb5425deebd6756
@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol	35fb271561f3dc72e91b3a42c6e40c2bb2e788cd8ca58014ac43f6198b8d32ca
@openzeppelin/contracts-upgradeable/utils/CountersUpgradeable.sol	5c1ac829a429b0c2ca9b4c9ed8b78d412320e9175e45f088c4e9056ef95fbf21
@openzeppelin/contracts/security/ReentrancyGuard.sol	aa73590d5265031c5bb64b5c0e7f84c44cf5f8539e6d8606b763adac784e8b2e
@openzeppelin/contracts/token/ERC20/IERC20.sol	94f23e4af51a18c2269b355b8c7cf4db8003d075c9c541019eb8dcf4122864d5
contracts/vestor.sol	8e96043137567458e08a12ac5e0f4609d31cf84823f1c639fc120a8e5127d805

Introduction

The Vestor contract is responsible for keeping the vesting amount and sharing it with the investors proportionally to the time that has elapsed.

Roles

Any user can execute the claim function but the claim function provides the vesting funds only to the beneficial addresses.

Contract Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	TKI	Token Vesting Issue	Unresolved
●	RM	Redundant Mapping	Unresolved
●	STC	Succeeded Transfer Check	Unresolved
●	CO	Code Optimization	Unresolved
●	MC	Missing Check	Unresolved
●	L01	Public Function could be Declared External	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L11	Unnecessary Boolean equality	Unresolved
●	L13	Divide before Multiply Operation	Unresolved

TVI - Token Vesting Issue

Criticality	critical
Location	contract.sol#L135
Status	Unresolved

Description

The vesting functionality may produce some vulnerabilities during the claiming period.

Full Unlock

If a user claims the vested token when the vesting period is elapsed no token will be transferred.

For example, supposed we have a user with address Y that has vested X amount of tokens.

- amountofinvestorsbyindex[Y][_contractid] = X
- gettotalamountunlocked(Y) = X

```
uint256 deduct = amountofinvestorsbyindex[ _address][_contractid] -
gettotalamountunlocked(_address); // X - X = 0

amountofinvestorsbyindex[ _address][_contractid] = deduct; // 0

IERC20(c._tokencontractaddress).transfer(_address,gettotalamountunlocked(_addres
s)); // gettotalamountunlocked(_address) = 0
```

As a result zero amount will be transferred to the user. Because the gettotalamountunlocked method will yield zero.

Partial Unlock

If a user claims the vested tokens sequentially before the end of the vesting period, then he will not be able to claim the remaining amount.

For instance, suppose that a user:

1. Claim tokens at timestamp x, the time elapsed 50% of vesting period from initial vest.
2. Claim tokens at timestamp y, the time elapsed 60% of vesting period from timestamp x

As a result, the following expression will underflow and the user will never be able to claim the remaining amount.

```
amountofinvestorsbyindex[ _address][_contractid] - gettotalamountunlocked(_address);
```

```
function claimtokens(uint256 _contractid,address _address)public nonReentrant  {
        vest storage c = vestcontracts[_contractid];

   require(isWhitelisted( _address,_contractid)!=false,"you are not eligible for the claim");
   require(block.timestamp >= c._startPeriod , "vesting has not yet began");
   require(amountofinvestorsbyindex[ _address][_contractid] > 0,"all of your tokens have been
claimed");
   uint256 deduct = amountofinvestorsbyindex[ _address][_contractid] -
gettotalamountunlocked(_address); // 50 - 60
   amountofinvestorsbyindex[ _address][_contractid] = deduct;
   IERC20(c._tokencontractaddress).transfer(_address,gettotalamountunlocked(_address));
   userclaimingstart[ _address][_contractid] = block.timestamp;
   emit HTLCERC20Withdraw( _address,_contractid,gettotalamountunlocked(_address));
}
```

Recommendation

The team is advised to carefully check if the implementation follows the expected business logic.

RM - Redundant Mapping

Criticality	critical
Location	contract.sol
Status	Unresolved

Description

The contract is using a contract counter in order to determin the vesting contract order. Even the methods accept the contract counter variable, internally they access merely the first index. Additionally, since the cloned contracts are initialized as minimal proxies, they are not upgradeable. As a result, the `CountersUpgradeable.Counter private contractid;` and all the contract counter mappings are redundant.

```
function gettime (address _address)internal view returns(uint256){
    uint256 total = block.timestamp - userclaimingstart[ _address][0];
    return total;
}
```

Hence, all the mappings from id to address should be flattened and the methods should erase the contractid argument.

```
mapping(address =>mapping(uint256 =>uint256))public amountofinvestorsbyindex;
mapping(address =>mapping(uint256 => uint256))public userclaimingstart;
mapping(address =>mapping(uint256 => uint256))public timesclaimablebyinvestors;

function claimtokens(uint256 _contractid,address _address)public nonReentrant  {
// ...
}
...
```

Recommendation

It is recommended to implement a version that does not use the multiple contracts logic.

```
mapping(address => uint256)public amountofinvestorsbyindex;
mapping(address => uint256)public userclaimingstart;
mapping(address => uint256)public timesclaimablebyinvestors;

function claimtokens(address _address)public nonReentrant  {
// ...
}
...
```

STC - Succeeded Transfer Check

Criticality	minor / informative
Location	contract.sol#L135
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(c._tokencontractaddress).transfer(_address,gettotalamountunlocked(_address));
```

Recommendation

The contract should check if the result of the transfer methods is successful.

CO - Code Optimization

Criticality	minor / informative
Location	contract.sol#L127,125,162,187,209
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.

Precondition

The contract checks if an address is whitelisted for a claiming. A user that is applicable to claim tokens is also checked by the amountofinvestorsbyindex mapping. Since, these two preconditions create a tautology, one of them could be eliminated.

```
function claimtokens(uint256 _contractid,address _address)public nonReentrant  {
        vest storage c = vestcontracts[_contractid];

    require(isWhitelisted( _address,_contractid)!=false,"you are not eligible for the claim");
    require(block.timestamp >= c._startPeriod , "vesting has not yet began");
    require(amountofinvestorsbyindex[ _address][_contractid] > 0,"all of your tokens have been
claimed");
```

Storage keyword

Wrong utilization of storage keyword in multiple sections. The storage keyword should not be used in view methods and case where the variable is not indented be used as a state variable.

```
vest storage c = vestcontracts[_contractid];
```

The contract could avoid executing code segments when it is not required. For instance, if the first require statement fails, then the initial assignment will be executed unnecessarily.

```
vest storage c = vestcontracts[_contractid];

require(isWhitelisted( _address,_contractid)!=false,"you are not eligible for
the claim");
require(block.timestamp >= c._startPeriod , "vesting has not yet began");
```

Recommendation

The contract could rewrite some code segments so the runtime will be more performant.

The function `isWhitelisted` could be removed from the implementation.

It is recommended to remove storage keywords from variables that are not used as storage. For instance `vest c = vestcontracts[_contractid];`

MC - Missing Check

Criticality	minor / informative
Location	contract.sol#L43
Status	Unresolved

Description

The contract is processing variables that have not properly sanitized and checked that they form the proper shape. These variables may produce vulnerability issues. To be more specific the variable `startperiod` and `cliffperiod` are not properly sanitized

```
function initialize(
    string memory name,
    address tokencontractaddress,
    address[]calldata investors,
    uint256 vestingPeriod,
    uint256[]calldata amountperinvestors,
    uint256 startperiod, uint256 cliffperiod
    ) initializer public {
       vestTokens(
        name,
        tokencontractaddress,
        investors,
        vestingPeriod,
        amountperinvestors,
        startperiod,
        cliffperiod
        );
   }
```

The method `gettime` could underflow and revert. If the variable `userclaimingstart[_address][0]` is greater than the block.timestamp.

```
function gettime(address _address) internal view returns (uint256) {
    uint256 total = block.timestamp - userclaimingstart[_address][0];
    return total;
  }
```

The contract does check if it has the available total balance to start the vesting.

```
int256 totalamount = addforinvestors(amountperinvestors,vestingPeriod,cliffperiod);
```

Recommendation

The contract should properly check the variables according to the required specifications.

- Startperiod should be greater than zero.
- Cliffperiod should be greater than zero.
- Block.timestamp should be greater than userclaimingstart[_address][0].
- The contract should check if it has the totalamount in tokens.

L01 - Public Function could be Declared External

Criticality	minor / informative
Location	contracts/vestor.sol#L124,234,43,244,147,278
Status	Unresolved

Description

Public functions that are never called by the contract should be declared external to save gas.

```
claimtokens
fetchcontractswhitelisted
initialize
getamount
getContract
addforamount
```

Recommendation

Use the external attribute for functions never called from the contract.

L04 - Conformance to Solidity Naming Conventions

Criticality	minor / informative
Location	contracts/vestor.sol#L147,291,124,186,196,244,207,23,278,234,10,222
Status	Unresolved

Description

Solidity defines a naming convention that should be followed. Rule exceptions:

- Allow constant variable name/symbol/decimals to be lowercase.
- Allow _ at the beginning of the mixed_case match for private variables and unused parameters.

```
_contractId
_numbers
_address
_Contractid
_vestingperiod
vest
_user
_id
_contractid
...
```

Recommendation

Follow the Solidity naming convention.
https://docs.soliditylang.org/en/v0.4.25/style-guide.html#naming-conventions.

L11 - Unnecessary Boolean equality

Criticality	minor / informative
Location	contracts/vestor.sol#L147,124
Status	Unresolved

Description

The comparison to boolean constants is redundant. Boolean constants can be used directly and do not need to be compared to true or false.

```
haveContract(_contractId) == false
require(bool,string)(isWhitelisted(_address,_contractid) != false,you are not eligible for the claim)
```

Recommendation

Remove the equality to the boolean constant.

L13 - Divide before Multiply Operation

Criticality	minor / informative
Location	contracts/vestor.sol#L207
Status	Unresolved

Description

Performing divisions before multiplications may cause lose of prediction.

```
total = timesclaimablebyinvestors[_address][0] * (gettime(_address) / c.cliffperiod)
```

Recommendation

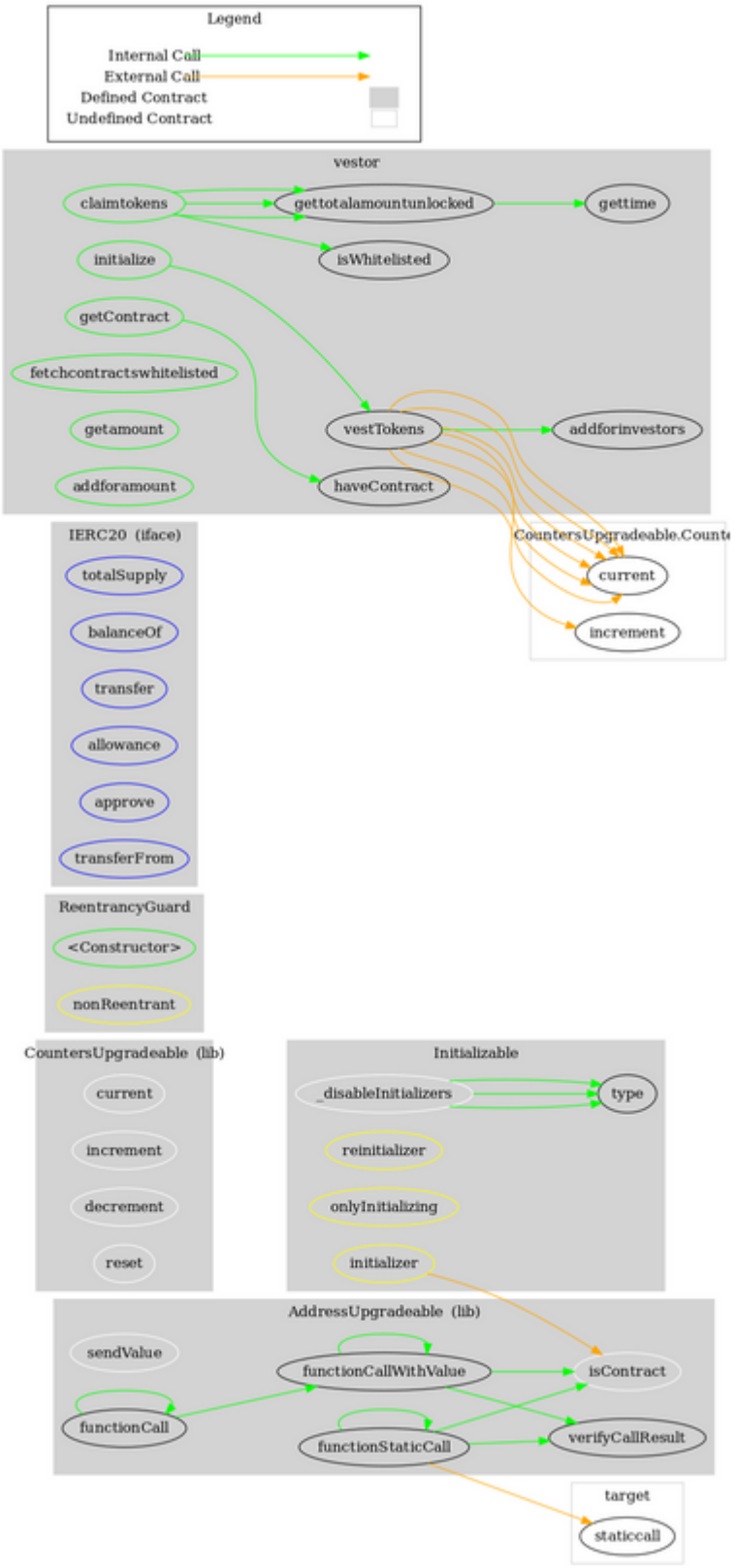
The multiplications should be prior to the divisions.

Contract Functions

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Initializable	Implementation			
	_disableInitializers	Internal	✓	
AddressUpgradeable	Library			
	isContract	Internal		
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionStaticCall	Internal		
	functionStaticCall	Internal		
	verifyCallResult	Internal		
CountersUpgradeable	Library			
	current	Internal		
	increment	Internal	✓	
	decrement	Internal	✓	
	reset	Internal	✓	
ReentrancyGuard	Implementation			
	<Constructor>	Public	✓	-
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-

	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
vestor	Implementation	Initializable, Reentrancy Guard		
	initialize	Public	✓	initializer
	vestTokens	Internal	✓	
	claimtokens	Public	✓	nonReentrant
	getContract	Public		-
	isWhitelisted	Public		-
	gettime	Internal		
	gettotalamountunlocked	Public		-
	haveContract	Internal		
	fetchcontractswhitelisted	Public		-
	getamount	Public		-
	addforamount	Public		-
	addforinvestors	Public		-

Contract Flow



Summary

The Vestor contract implements a tokens vestor mechanism. This audit investigates security issues, mentions business logic concerns, and potential improvements.

Disclaimer

All the content provided in this document is for general information only and should not be used as financial advice or a reason to buy any investment.

Cyberscope team provides no guarantees against the sale of team tokens or the removal of liquidity by the project audited in this document. Always Do your own research and protect yourselves from being scammed.

The Cyberscope team has audited this project for general information and only expresses their opinion based on similar projects and checks from popular diagnostic tools. Under no circumstances did Cyberscope receive a payment to manipulate those results or change the awarding badge that we will be adding in our website.

Always Do your own research and protect yourselves from scams. This document should not be presented as a reason to buy or not buy any particular token.

The Cyberscope team disclaims any liability for the resulting losses.

About Cyberscope

Coinscope audit and K.Y.C. service has been rebranded to Cyberscope.

Coinscope is the leading early coin listing, voting and auditing authority firm. The audit process is analyzing and monitoring many aspects of the project. That way, it gives the community a good sense of security using an informative report and a generic score.

Cyberscope and Coinscope are aiming to make crypto discoverable and efficient globally. They provide all the essential tools to assist users draw their own conclusions.

The Cyberscope team

https://www.cyberscope.io