Cyberscope

Audit Report

Openmesh Vesting

July 2024

Repository	https://github.com/Plopmenz/vesting/
Commit	845065fd726b25695f4b2170817345bfeee2fe55
Audited by	© cyberscope

Table of Contents

Review

Repository	https://github.com/Plopmenz/vesting/
Commit	845065fd726b25695f4b2170817345bfeee2fe55

Audit Updates

Initial Audit	02 Jul 2024

Source Files

Filename	SHA256
vesting-main/src/SingleBeneficiaryLinearERC20TransferVestingStoppable.sol	72eb17ae275653b586e5293b51c1c62611ba4e5d1ecb10afd99b80724ef841bf
vesting-main/src/SingleBeneficiaryLinearERC20TransferVesting.sol	c1db399cb7e76b537c84a22456527ed4197f4b6ff2283872cade934865b9ec35
vesting-main/src/SingleBeneficiaryLinearERC20MintVesting.sol	2d967a6f3af24be3e5faf018f433d972799f6ef6220d9d7165afb03fb28098e8
vesting-main/src/MultiERC721TokenLinearERC20TransferVesting.sol	9df6531fa9b3c82a5aef9e334212af93a0ef3b34049c7aa65bc534ecf5f0ce84
vesting-main/src/MultiERC721TokenLinearERC20MintVesting.sol	4e1f9c34561b59a40301caf5a803226f17fb7e96cb28ed65d1be46a34f45f5d5
vesting-main/src/vesting/Vesting.sol	31c4a499938363f68976ae818f06b44563ca888249defb88e32df733bc544a2d
vesting-main/src/vesting/TokenReleased.sol	3c2ea37c4ccba1c210870ec0f769ba595269e82bd2766abd708e2c05a8138d6a
vesting-main/src/vesting/TokenBeneficiary.sol	c5a47ac74bb303c9fc1bb833890c90e0f6f27ed7893d2a52d18034715ada7db7

vesting-main/src/vesting/Released.sol	9408df5054e7bce5775b9fd6fe6241730 5549073d8f7cccfd830e2692e714c89
vesting-main/src/vesting/Manager.sol	8fb7320714b8d7df26503e5c0430dc004 45844784e17b1a395324a6abb51054d
vesting-main/src/vesting/Managed.sol	bd3bd6a193cc8fe0481777839bbf2c137 8711974f16a5e0397c3f65f9ef6ceee
vesting-main/src/vesting/LinearVestingStoppable.sol	b1d2252c8046f0942f09ea426dc70b0b5 10c0e1d3b8fec29e77f661510dc3ffb
vesting-main/src/vesting/LinearVesting.sol	cfaabc477e54efbf85b3544fff8afd2f82fb e4fd78c3be54f02c6de62cca6333
vesting-main/src/vesting/ERC721OwnerBeneficiary.sol	d91f4b5dd8f7c6df2ba9878c6ebafe94c6 53d740327f489edc7d9f4da89faf8e
vesting-main/src/vesting/Beneficiary.sol	6ccd3025499be41c11aa6e234e95c5e3 a7d4ad369db1e8cff25c6d74192b88e8
vesting-main/src/vesting/AddressReleased.sol	ced9b0acd3f67852dac04511503e6477 c99868974734d901599274681244f7db
vesting-main/src/vesting/extensions/SingleBeneficiaryLinearVesting.sol	04184d12d06e87a77fa8b10cda40460fa 972c04a090a2455a5a1d3e2ae7e744d
vesting-main/src/vesting/extensions/MultiTokenLinearVesting.sol	703e57fb035a789313313065b9d42e43 56ea6dd64b419579379d00d4bbdd146 6
vesting-main/src/storage/TokenReleasedStorage.sol	aa8b4ca4752409dc709bcab847606c0a 19af4c335311b8ae3117ea3840a8786a
vesting-main/src/storage/ReleasedStorage.sol	d754208ae994186bf9d7c17123b340cc 35902ecdd9692368610e5acb5f8e543b
vesting-main/src/storage/ManagerStorage.sol	8ab10b4397f61391958e2330c4ea288c ac73a7cd26b7825261dfba2c4174e578
vesting-main/src/storage/LinearVestingStorage.sol	c5cab470e94d5942667a3801c62f1769 8afba125fe81cd453a22ce4f9b2eaf4d

vesting-main/src/storage/ERC721OwnerBeneficiaryStorage.sol	01e6353711546c5ba7639b3f37673358473491dd4f0a781260e82726eed32585
vesting-main/src/storage/ERC20RewardStorage.sol	f579704cff8c02c901940b9d927f8eed1ea77ec1f581f29da146d8a2b87d8d29
vesting-main/src/storage/BeneficiaryStorage.sol	0bf71e98a375404e29ed9ef1df46e69e653b7b4a07103a2e7c67c3588b423141
vesting-main/src/storage/AddressReleasedStorage.sol	d26088bd276c2c7b73fc1426f4a0e7d2f11041b9008aad24d39fa97bab8e71f4
vesting-main/src/rewards/Reward.sol	f00a73f266a50423eb039fd36a11179032cb5bee9bc18eab0cb9da5892d424e1
vesting-main/src/rewards/IERC20Mintable.sol	2150b44fca3f1f2c8b15dcd27f48230a4887af604e2373ed1e4f95e7e2a7023c
vesting-main/src/rewards/ERC20TransferReward.sol	6dbb0665487e42ef0b9c34f7cb482d52226a2196c3a930dc8fc2eecb16b6b5e2
vesting-main/src/rewards/ERC20MintReward.sol	dd656b8a653f5870c0c934b74363594dd93bdc37f87e0e660a3d11a67efde331
vesting-main/src/managed/JITSingleBeneficiaryLinearERC20TransferVestingStoppableManager.sol	828354b21a915b0cf0bbc76de92199ec7020d78a3f08ea7a01ffcb46b6037caa
vesting-main/src/managed/JITSingleBeneficiaryLinearERC20TransferVestingManager.sol	c6c5d639aef58e52f2b48e6a381f5c531f14a40ff87012e62b42a1d9348bddae

Overview

JITSingleBeneficiaryLinearERC20TransferVestingManager Contract

The `JITSingleBeneficiaryLinearERC20TransferVestingManager` is a smart contract designed to efficiently manage the vesting of ERC20 tokens for individual beneficiaries. This contract provides a secure and automated way to distribute tokens over a specified period, ensuring that beneficiaries receive their tokens in a controlled and predictable manner.

The contract begins by initializing with an immutable reference to an ERC20 token and a vesting proxy implementation. This setup ensures that the token and the vesting logic are predefined and cannot be altered once the contract is deployed, providing stability and security to the vesting process.

The primary functionality of the contract allows the owner to create new vesting schedules for beneficiaries. Each vesting schedule is defined by the amount of tokens, the start time, the duration of the vesting period, and the beneficiary's address. The contract generates a deterministic address for each vesting schedule, ensuring that the same schedule cannot be created multiple times for the same beneficiary. Once a vesting schedule is created, it is initialized with the specified parameters, and an event is emitted to signal the creation of the new vesting contract.

A key feature of the contract is its ability to calculate the address of a vesting contract without deploying it. This predictive capability allows for efficient management and verification of vesting schedules, ensuring that each schedule is unique and correctly configured before deployment.

The contract also facilitates the release of vested tokens. At any point, anyone can trigger the release of vested tokens for a beneficiary. The contract calculates the amount of tokens that have vested but not yet been released, mints the required tokens, and transfers them to the beneficiary. This process ensures that tokens are only created when they are due to be distributed, maintaining an efficient token supply.

Additional Functionality from Associated Contracts

The associated vesting contract manages the detailed vesting logic, including tracking the amount of tokens released and calculating the amount that can be released at any given time. The vesting contract ensures that tokens are vested linearly over the specified duration, starting from the defined start time. The contract updates the released amount and transfers the vested tokens to the beneficiary as they become available.

The `JITSingleBeneficiaryLinearERC20TransferVestingManager` ensures that no unvested tokens are minted and tokens are only minted upon claim. This method maintains an efficient and clean token supply, with the contract allowing the owner to create multiple vesting schedules. The deterministic creation of vesting contract addresses prevents duplicate schedules and allows for predictable management. Each vesting contract cannot independently mint tokens, avoiding clutter in token access control permissions. The linear vesting process guarantees that once tokens are vested, they cannot be unvested unless the minting permission is revoked from the manager contract. Additionally, anyone can pay the gas fees to claim vested tokens on behalf of the beneficiary, ensuring timely distribution of rewards. This contract is particularly useful for scenarios where rewards must be granted to beneficiaries under any circumstances, such as compensation for actions performed.

JITSingleBeneficiaryLinearERC20TransferVestingStoppableManager Contract

The `JITSingleBeneficiaryLinearERC20TransferVestingStoppableManager` contract has similar functionality to the `JITSingleBeneficiaryLinearERC20TransferVestingManager` contract, as both are designed to manage the linear vesting of ERC20 tokens. Both contracts handle the creation, management, and release of vested tokens for individual beneficiaries in a controlled and predictable manner.

Stopping Vesting Schedules

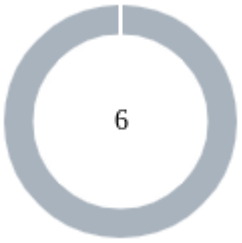
The key feature that distinguishes the `JITSingleBeneficiaryLinearERC20TransferVestingStoppableManager` contract is its ability to stop ongoing vesting schedules. This new functionality allows the owner to halt vesting before the initially specified duration elapses. The owner can specify a new duration, provided it is in the future and shorter than the original duration, ensuring that vesting adjustments are proactive and do not affect past vesting periods.

This stopping functionality is particularly useful for scenarios where vesting should continue only while certain conditions are met. For example, if vesting is provided to core contributors of a project, it can be stopped if they cease contributing. The contract ensures that tokens are only vested while the predefined conditions are satisfied, offering greater flexibility and control over the distribution process.

The contract uses a deterministic method to generate unique addresses for each vesting schedule, ensuring that identical schedules cannot be created multiple times. It facilitates the release of vested tokens by calculating the releasable amount, minting the required tokens, and transferring them to the beneficiary.

In summary, the `JITSingleBeneficiaryLinearERC20TransferVestingStoppableManager` contract extends the functionality of the `JITSingleBeneficiaryLinearERC20TransferVestingManager` contract, by introducing the ability to stop vesting schedules, offering enhanced control and flexibility for managing token vesting based on changing conditions.

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	6

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	6	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	MDV	Missing Deployment Verification	Unresolved
	MIV	Missing Input Validations	Unresolved
	RFC	Redundant Function Calls	Unresolved
	TMO	Token Minting Optimization	Unresolved
	L04	Conformance to Solidity Naming Conventions	Unresolved
	L19	Stable Compiler Version	Unresolved

MDV - Missing Deployment Verification

Criticality	Minor / Informative
Location	vesting-main/src/SingleBeneficiaryLinearERC20TransferVesting.sol#L34,48
Status	Unresolved

Description

The contract contains functions that rely on the presence of a deployed contract at a computed address but does not include necessary checks to verify this deployment. Functions like `createVesting` and `release` perform operations predicated on the assumption that the addressed vesting contracts were successfully deployed and initialized. However, without a verification mechanism, these functions might interact with addresses where no contract is present, leading to failed operations and a lack of clarity in error handling.

```
    function createVesting(uint128 _amount, uint64 _start, uint64
_duration, address _beneficiary)
        external
        onlyOwner
        returns (address vesting)
    {
        vesting = Clones.cloneDeterministic(
            address(implementation), keccak256(abi.encodePacked(_amount,
_start, _duration, _beneficiary))
        );

SingleBeneficiaryLinearERC20TransferVestingProxy(vesting).initialize(
            token, _amount, _start, _duration, _beneficiary
        );
        emit VestingCreated(vesting, _amount, _start, _duration,
_beneficiary);
    }

    function release(uint128 _amount, uint64 _start, uint64 _duration,
address _beneficiary)
        external
        returns (uint256 released)
    {
        SingleBeneficiaryLinearERC20TransferVesting vesting =

SingleBeneficiaryLinearERC20TransferVesting(getAddress(_amount, _start,
_duration, _beneficiary));
        released = vesting.releasable();
        token.mint(address(vesting), released);
        vesting.release();
    }
```

Recommendation

It is recommended to incorporate a `require` check in functions that interact with vesting contracts to ensure that these addresses correspond to deployed contracts. This verification should utilize a method to check for contract code at the specified address, reverting with a clear error message if no contract is found. This adjustment will safeguard the integrity of contract interactions, enhance error transparency, and prevent operations on non-existent contracts, thereby boosting overall system reliability and user trust.

MIV - Missing Input Validations

Criticality	Minor / Informative
Location	vesting-main/src/SingleBeneficiaryLinearERC20TransferVesting.sol#L34
Status	Unresolved

Description

The contract is currently implemented without crucial validation checks in its `createVesting` function, potentially leading to inconsistent and undesirable outcomes. Specifically, the function lacks verification to ensure that the vesting amount (`_amount`) is greater than zero, the vesting start time (`_start`) is set to a future date (beyond the current block timestamp), and the designated beneficiary (`_beneficiary`) is not an uninitialized or zero address. The absence of these checks could result in the creation of vesting schedules that are either ineffective or directly violate intended constraints. For example, creating a vesting schedule with a zero or negative amount, or vesting to non-existent addresses, or immediate vesting due to a past start time are all scenarios that could occur without these validations.

```
    function createVesting(uint128 _amount, uint64 _start, uint64
_duration, address _beneficiary)
        external
        onlyOwner
        returns (address vesting)
    {
        vesting = Clones.cloneDeterministic(
            address(implementation),
keccak256(abi.encodePacked(_amount, _start, _duration,
_beneficiary))
        );

SingleBeneficiaryLinearERC20TransferVestingProxy(vesting).initia
lize(
            token, _amount, _start, _duration, _beneficiary
        );
        emit VestingCreated(vesting, _amount, _start, _duration,
_beneficiary);
    }
```

Recommendation

It is recommended to reconsider the intended functionality and add checks to the `createVesting` function. These should include verifying that the `_amount` is greater than zero to prevent the creation of meaningless vesting schedules. Additionally, ensuring that the `_start` time is strictly greater than the current block timestamp will guarantee that vesting schedules are future-oriented and do not start in the past. Finally, verifying that the `_beneficiary` address is not the zero address will prevent the misallocation of tokens to invalid addresses. Implementing these safeguards will enhance the robustness and reliability of the vesting process, aligning it more closely with standard security practices in smart contract development.

RFC - Redundant Function Calls

Criticality	Minor / Informative
Location	vesting-main/src/SingleBeneficiaryLinearERC20TransferVesting.sol#L48 vesting-main/src/vesting/TokenReleased.sol#L12,17 vesting-main/src/vesting/extensions/SingleBeneficiaryLinearVesting.sol#L20 vesting-main/src/vesting/Released.sol#L20
Status	Unresolved

Description

The contract is currently structured in a way that causes the `releasable` function to be calculated twice during the execution of the `release` function. Initially, `releasable` is called to determine the amount of tokens that can be released, and this amount is then minted to the vesting contract. Subsequently, the `release` function of the vesting contract is invoked, which internally calls the `releasable` function again to determine the amount to release. This redundant computation not only wastes gas but also complicates the logic, potentially leading to inconsistencies in the management of the release process.

```
   function release(uint128 _amount, uint64 _start, uint64
_duration, address _beneficiary)
        external
        returns (uint256 released)
    {
        SingleBeneficiaryLinearERC20TransferVesting vesting =

SingleBeneficiaryLinearERC20TransferVesting(getAddress(_amount,
_start, _duration, _beneficiary));
        released = vesting.releasable();
        token.mint(address(vesting), released);
        vesting.release();
    }

    function released(uint256 _tokenId) public view virtual
returns (uint256) {
        TokenReleasedStorage.Storage storage $ =
TokenReleasedStorage.getStorage();
        return $.released[_tokenId];
    }

    function releasable(uint256 _tokenId) public view virtual
returns (uint256) {
        return _vestingUnlocked() - released(_tokenId);
    }

    ...
    function release() public virtual {
        _release(beneficiary());
    }
    ...
    function _release(address _account) internal virtual {
        uint256 releaseAmount = releasable();
        ReleasedStorage.Storage storage $ =
ReleasedStorage.getStorage();
        $.released += releaseAmount;
        _reward(_account, releaseAmount);
    }
```

Recommendation

It is recommended to reconsider and modify the functionality to only call the `releasable` function once within the release process. By ensuring that the releasable amount is computed once and used consistently throughout the function's execution, the contract can achieve a more efficient and error-resistant implementation. This change would reduce the

computational overhead and potential for errors, streamlining the release process and conserving network resources.

TMO - Token Minting Optimization

Criticality	Minor / Informative
Location	vesting-main/src/SingleBeneficiaryLinearERC20TransferVesting.sol#L48 vesting-main/src/vesting/TokenReleased.sol#L12,17 vesting-main/src/vesting/extensions/SingleBeneficiaryLinearVesting.sol#L20 vesting-main/src/vesting/Released.sol#L20
Status	Unresolved

Description

The contract is designed to mint tokens to a vesting contract, which subsequently handles the transfer of these tokens to the beneficiary through the `release` function. This two-step process involves minting tokens to the vesting contract's address initially, and then executing an internal transfer to the beneficiary. This approach, while maintaining a separation of concerns between token minting and distribution, incurs additional gas costs due to the two separate transaction steps, minting to the vesting contract and then transferring to the beneficiary.

```
   function release(uint128 _amount, uint64 _start, uint64
_duration, address _beneficiary)
        external
        returns (uint256 released)
    {
        SingleBeneficiaryLinearERC20TransferVesting vesting =

SingleBeneficiaryLinearERC20TransferVesting(getAddress(_amount,
_start, _duration, _beneficiary));
        released = vesting.releasable();
        token.mint(address(vesting), released);
        vesting.release();
    }

    function released(uint256 _tokenId) public view virtual
returns (uint256) {
        TokenReleasedStorage.Storage storage $ =
TokenReleasedStorage.getStorage();
        return $.released[_tokenId];
    }

    function releasable(uint256 _tokenId) public view virtual
returns (uint256) {
        return _vestingUnlocked() - released(_tokenId);
    }

    ...
    function release() public virtual {
        _release(beneficiary());
    }
    ...
    function _release(address _account) internal virtual {
        uint256 releaseAmount = releasable();
        ReleasedStorage.Storage storage $ =
ReleasedStorage.getStorage();
        $.released += releaseAmount;
        _reward(_account, releaseAmount);
    }
```

Recommendation

It is recommended to consider directly minting the released amount to the beneficiary's address to save gas. By eliminating the intermediary step of minting to the vesting contract, the contract can reduce the number of transactions required and the overall gas consumption. This modification not only streamlines the process but also enhances the

efficiency of the token distribution mechanism. Implementing this change would optimize the contract's operations and potentially lower the cost burden on users.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	vesting-main/src/vesting/Released.sol#L9 vesting-main/src/vesting/Manager.sol#L11 vesting-main/src/vesting/LinearVesting.sol#L9,10,11,12 vesting-main/src/vesting/extensions/SingleBeneficiaryLinearVesting.sol#L9,10,11,12,13 vesting-main/src/vesting/Beneficiary.sol#L7 vesting-main/src/SingleBeneficiaryLinearERC20TransferVestingStoppable.sol#L15,16,17,18,19,20,21,54,55,56,57,58,59 vesting-main/src/SingleBeneficiaryLinearERC20TransferVesting.sol#L10,11,12,13,14,15,36 vesting-main/src/rewards/ERC20TransferReward.sol#L12 vesting-main/src/managed/JITSingleBeneficiaryLinearERC20TransferVestingStoppableManager.sol#L31,32,33,34,44,45,46,47,66,67,68,69,80,81,82,83,84 vesting-main/src/managed/JITSingleBeneficiaryLinearERC20TransferVestingManager.sol#L30,31,32,33,43,44,45,46,63,64,65,66
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 __Released_init() internal {}

function __Manager_init(address _manager) internal {
        ManagerStorage.Storage storage $ =
ManagerStorage.getStorage();
        $.manager = _manager;
    }
address _manager

...
```

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.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	vesting-main/src/vesting/Vesting.sol#L2 vesting-main/src/vesting/Released.sol#L2 vesting-main/src/vesting/Manager.sol#L2 vesting-main/src/vesting/Managed.sol#L2 vesting-main/src/vesting/LinearVestingStoppable.sol#L2 vesting-main/src/vesting/LinearVesting.sol#L2 vesting-main/src/vesting/extensions/SingleBeneficiaryLinearVesting.sol#L2 vesting-main/src/vesting/Beneficiary.sol#L2 vesting-main/src/storage/ReleasedStorage.sol#L2 vesting-main/src/storage/ManagerStorage.sol#L2 vesting-main/src/storage/LinearVestingStorage.sol#L2 vesting-main/src/storage/ERC20RewardStorage.sol#L2 vesting-main/src/storage/BeneficiaryStorage.sol#L2 vesting-main/src/SingleBeneficiaryLinearERC20TransferVestingStoppable.sol#L2 vesting-main/src/SingleBeneficiaryLinearERC20TransferVesting.sol#L2 vesting-main/src/rewards/Reward.sol#L2 vesting-main/src/rewards/ERC20TransferReward.sol#L2 vesting-main/src/managed/JITSingleBeneficiaryLinearERC20TransferVestingStoppableManager.sol#L2 vesting-main/src/managed/JITSingleBeneficiaryLinearERC20TransferVestingManager.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 ^0.8.0;
```

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.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
SimpleImport	Implementation			
SingleBeneficiaryLinearERC20TransferVestingTest	Implementation	Test		
	setUp	Public	✓	-
	getVesting	Internal	✓	
	test_releasable	Public	✓	-
	test_release	Public	✓	-
	test_beforeStart	Public	✓	-
	test_init	Public	✓	-
SingleBeneficiaryLinearERC20MintVestingTest	Implementation	Test		
	setUp	Public	✓	-
	getVesting	Internal	✓	
	test_linearVesting	Public	✓	-
	test_release	Public	✓	-
	test_beforeStart	Public	✓	-
	test_init	Public	✓	-

MultiERC721TokenLinearERC20TransferVestingTest	Implementation	Test		
	setUp	Public	✓	-
	getVesting	Internal	✓	
	test_linearVesting	Public	✓	-
	test_release	Public	✓	-
	test_beforeStart	Public	✓	-
	test_init	Public	✓	-
	test_beneficiary	Public	✓	-
MultiERC721TokenLinearERC20MintVestingTest	Implementation	Test		
	setUp	Public	✓	-
	getVesting	Internal	✓	
	test_linearVesting	Public	✓	-
	test_release	Public	✓	-
	test_beforeStart	Public	✓	-
	test_init	Public	✓	-
	test_beneficiary	Public	✓	-
JITSingleBeneficiaryLinearERC20TransferVestingStoppableManagerTest	Implementation	Test		

	setUp	Public	✓	-
	getVesting	Internal	✓	
	test_init	Public	✓	-
	test_release	Public	✓	-
	test_stop	Public	✓	-
	test_ownable	Public	✓	-
JITSingleBeneficiaryLinearERC20TransferVestingManagerTest	Implementation	Test		
	setUp	Public	✓	-
	getVesting	Internal	✓	
	test_init	Public	✓	-
	test_release	Public	✓	-
	test_ownable	Public	✓	-
ERC721Mock	Implementation	ERC721		
		Public	✓	ERC721
	mint	External	✓	-
ERC20Mock	Implementation	ERC20, IERC20Mintable		
		Public	✓	ERC20
	mint	External	✓	-

SingleBeneficiaryLinearERC20TransferVestingStoppable	Implementation	SingleBeneficiaryLinearERC20TransferVesting, Manager, LinearVestingStoppable		
	__SingleBeneficiaryLinearERC20TransferVestingStoppable_init	Internal	✓	
SingleBeneficiaryLinearERC20TransferVestingStoppableStandalone	Implementation	SingleBeneficiaryLinearERC20TransferVestingStoppable		
		Public	✓	-
SingleBeneficiaryLinearERC20TransferVestingStoppableProxy	Implementation	Initializable, SingleBeneficiaryLinearERC20TransferVestingStoppable		
		Public	✓	-
	initialize	External	✓	initializer
SingleBeneficiaryLinearERC20TransferVesting	Implementation	SingleBeneficiaryLinearVesting, ERC20TransferReward		
	__SingleBeneficiaryLinearERC20TransferVesting_init	Internal	✓	
SingleBeneficiaryLinearERC20TransferVestingStandalone	Implementation	SingleBeneficiaryLinearERC20TransferVesting		
		Public	✓	-

SingleBeneficiaryLinearERC20TransferVestingProxy	Implementation	Initializable, SingleBeneficiaryLinearERC20TransferVesting		
		Public	✓	-
	initialize	External	✓	initializer
SingleBeneficiaryLinearERC20MintVesting	Implementation	SingleBeneficiaryLinearVesting, ERC20MintReward		
	__SingleBeneficiaryLinearERC20MintVesting_init	Internal	✓	
SingleBeneficiaryLinearERC20MintVestingStandalone	Implementation	SingleBeneficiaryLinearERC20MintVesting		
		Public	✓	-
SingleBeneficiaryLinearERC20MintVestingProxy	Implementation	Initializable, SingleBeneficiaryLinearERC20MintVesting		
		Public	✓	-
	initialize	External	✓	initializer
MultiERC721TokenLinearERC20TransferVesting	Implementation	MultiTokenLinearVesting, ERC721OwnerBeneficiary, ERC20TransferReward		
	__MultiERC721TokenLinearERC20TransferVesting_init	Internal	✓	

MultiERC721TokenLinearERC20TransferVestingStandalone	Implementation	MultiERC721TokenLinearERC20TransferVesting		
		Public	✓	-
MultiERC721TokenLinearERC20TransferVestingProxy	Implementation	Initializable, MultiERC721TokenLinearERC20TransferVesting		
		Public	✓	-
	initialize	External	✓	initializer
MultiERC721TokenLinearERC20MintVesting	Implementation	MultiTokenLinearVesting, ERC721OwnerBeneficiary, ERC20MintReward		
	__MultiERC721TokenLinearERC20MintVesting_init	Internal	✓	
MultiERC721TokenLinearERC20MintVestingStandalone	Implementation	MultiERC721TokenLinearERC20MintVesting		
		Public	✓	-
MultiERC721TokenLinearERC20MintVestingProxy	Implementation	Initializable, MultiERC721TokenLinearERC20MintVesting		
		Public	✓	-
	initialize	External	✓	initializer

Vesting	Implementation	Reward		
	_vestingUnlocked	Internal		
TokenReleased	Implementation	Vesting, TokenBeneficiary		
	__TokenReleased_init	Internal	✓	
	released	Public		-
	releasable	Public		-
	_release	Internal	✓	
TokenBeneficiary	Implementation			
	beneficiary	Public		-
Released	Implementation	Vesting		
	__Released_init	Internal	✓	
	released	Public		-
	releasable	Public		-
	_release	Internal	✓	
Manager	Implementation	Managed		
	__Manager_init	Internal	✓	
	manager	Public		-

Managed	Implementation			
LinearVestingStoppable	Implementation	Managed		
	stop	Public	✓	onlyManager
LinearVesting	Implementation	Vesting		
	__LinearVesting_init	Internal	✓	
	amount	Public		-
	start	Public		-
	duration	Public		-
	_vestingUnlocked	Internal		
ERC721OwnerBeneficiary	Implementation	TokenBeneficiary		
	__ERC721OwnerBeneficiary_init	Internal	✓	
	ownerToken	Public		-
	beneficiary	Public		-
Beneficiary	Implementation			
	__Beneficiary_init	Internal	✓	
	beneficiary	Public		-
AddressReleased	Implementation	Vesting		
	__AddressReleased_init	Internal	✓	

	released	Public		-
	releasable	Public		-
	_release	Internal	✓	
SingleBeneficiaryLinearVesting	Implementation	Released, Beneficiary, LinearVesting		
	__SingleBeneficiaryLinearVesting_init	Internal	✓	
	release	Public	✓	-
MultiTokenLinearVesting	Implementation	TokenReleased, LinearVesting		
	__MultiTokenLinearVesting_init	Internal	✓	
	release	Public	✓	-
LinearVestingStorage	Library			
	getStorage	Internal		
ERC721OwnerBeneficiaryStorage	Library			
	getStorage	Internal		
ERC20RewardStorage	Library			
	getStorage	Internal		

BeneficiaryStorage	Library			
	getStorage	Internal		
AddressReleasedStorage	Library			
	getStorage	Internal		
Reward	Implementation			
	_reward	Internal	✓	
IERC20Mintable	Interface	IERC20		
	mint	External	✓	-
ERC20TransferReward	Implementation	Reward		
	__ERC20TransferReward_init	Internal	✓	
	token	Public		-
	_reward	Internal	✓	
ERC20MintReward	Implementation	Reward		
	__ERC20MintReward_init	Internal	✓	
	token	Public		-
	_reward	Internal	✓	

JITSingleBeneficiaryLinearERC20TransferVestingStoppableManager	Implementation	Ownable		
		Public	✓	Ownable
	getAddress	Public		-
	createVesting	External	✓	onlyOwner
	release	External	✓	-
	stop	External	✓	onlyOwner
JITSingleBeneficiaryLinearERC20TransferVestingManager	Implementation	Ownable		
		Public	✓	Ownable
	getAddress	Public		-
	createVesting	External	✓	onlyOwner
	release	External	✓	-

Summary

Openmesh contract implements a linear vesting mechanism for tokens. This audit investigates security issues, business logic concerns, and potential improvements to ensure efficient token distribution. The contract manages the creation, release, and stoppage of vesting schedules, providing flexibility and control over token vesting and distribution processes.

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