Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Hacken.IO Sample Token Vesting

Sample Audit Techspec

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Project Overview

Sample Solidity project is an ERC20 based vesting project. It allows its users (vestor) to deposit a specific amount of tokens to an existing payment plan (vesting period). The vesting admins create the vesting plans. . This project aims to solve xx and xx problems of the crypto vesting. To solve these problems above, Hacken.io Sample Vesting proposed the following approaches.
< The project's unique approach to vesting is described here. This description can be a high-level description. Its main purpose is to give auditors a basic idea of the concept >.

1. Functional Requirements

1.1. Roles

Hacken.IO samples project has two roles:

- **Vesting Admin:** All control regarding the vesting contract belongs to a vesting admin. A vesting admin can create payment plans, and in the case of necessity, can revoke a payment plan.
- **Vestor:** Vestor is the user of the project. A vestor can lock funds into open vesting plans and withdraw tokens from payment plans of which he/she is a participant.

1.2. Features

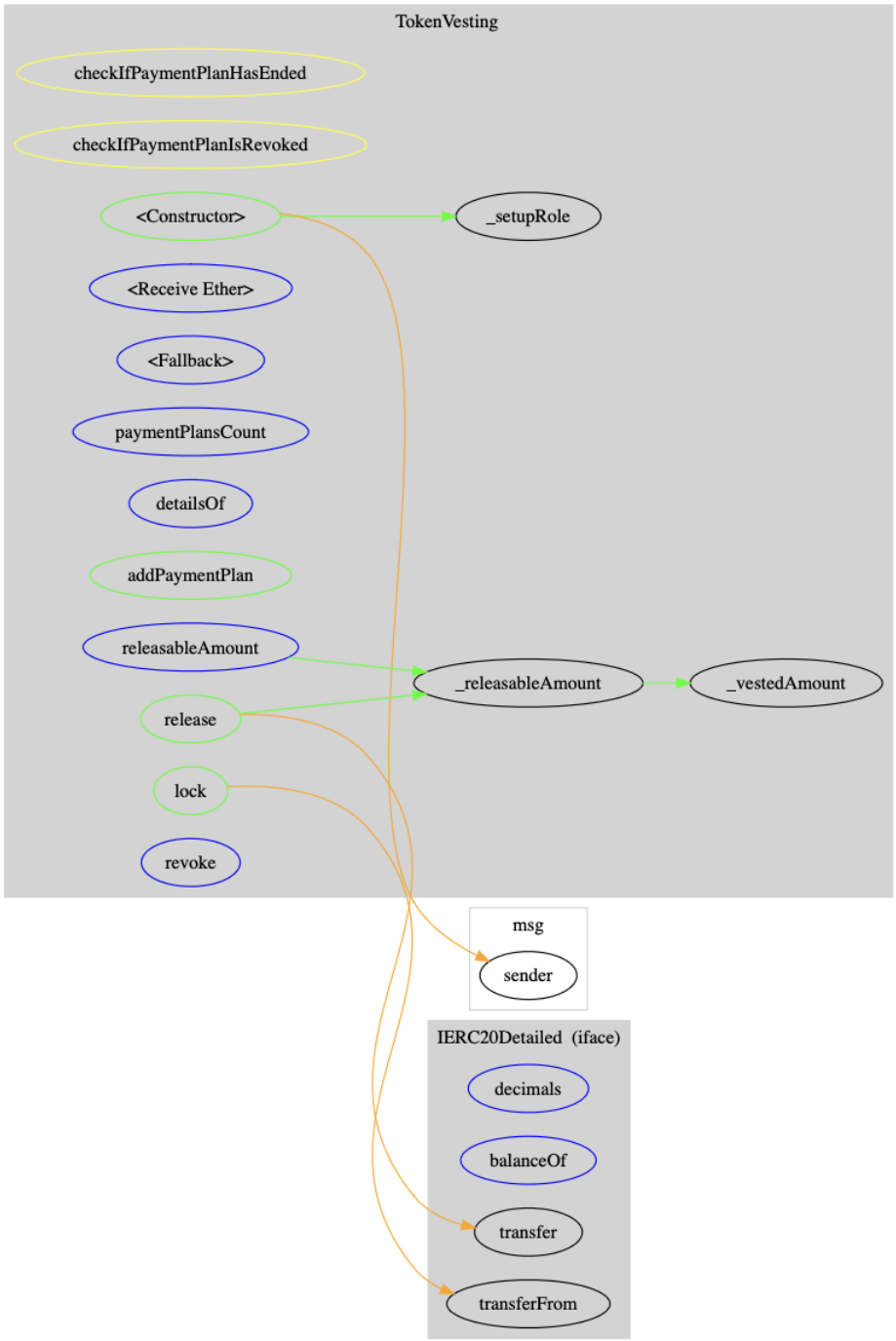
Hacken.IO Sample Token Vesting has the following features:

- Create new payment plans.
- Lock funds (tokens) into desired payment plans.
- Revoke payment plans.
- Release a vestor from a payment plan (a user withdraws his/her locked funds from a plan with rewards).
- Calculate a releasable amount for a vestor.
- Get detailed information about a vestor's participation in the payment plans.
- Get the number of payment plans.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

2. Technical Requirements

2.1. Architecture Overview



2.3. Contract Information

This section contains detailed information (their purpose, assets, functions, and events) about the contracts used in the project.

2.3.1. TokenVesting.sol

A token holder contract that can release its token balance gradually like a typical vesting scheme, with a cliff and vesting period. Optionally revocable by the admin.

2.3.1.1. Assets

Hacken.IO Sample Token Vesting contains two Structs:

- **PaymentPlan:** This object contains information about a payment plan.
  - uint256 periodLength - length of 1 period.
  - uint256 periods - total vesting periods.
  - uint256 cliffPeriods - number of periods that will be scipped.
  - bool revoked - True if plan is revoked, false otherwise.
- **Lock:** When a vestor locks funds in a payment plan. A *Lock* object is created. This object holds information about the vestor's participation in the payment plan.
  - address beneficiary - address of the beneficiary to whom vested tokens are transferred.
  - uint256 start - start the time (as Unix time), at which point vesting starts.
  - uint256 paymentPlan - payment plan to apply.
  - uint256 totalAmount - total amount in the lock.
  - uint256 released - the released amount of the lock (so far).

Besides the mentioned structs, the following entities are present in the project:

- **paymentPlans**: A public array of type Struct PaymentPlan holds the created payment plans of the Hacken.IO Sample Token Vesting.
- **lock:** An address <-> Lock object mapping. Aims to hold users' locks on the payment plans.
- **token:** TokenMock (ERC20 utility token) address.
- **PERCENT_100:** Constant denominator for %100.

2.3.1.2. Modifiers

Hacken.IO Sample Token Vesting has the following modifiers:

- **checkIfPaymentPlanHasEnded(PaymentPlan plan):** Checks if the given payment plan has been completed.
- **checkIfPaymentPlanIsRevoked(PaymentPlan plan):** Checks if the given payment plan has been revoked.

2.3.1.2. Functions

Hacken.IO Sample Token Vesting has the following functions:

- **constructor(IERC20Detailed _token):** Handles configurations and sets related addresses.
- **paymentPlansCount():** Returns number of payment plans created so far.
- **addPaymentPlan(uint256 periodLength, uint256 periods, uint256 cliffPeriods):** Allows a vesting admin to add a new payment plan.
- **lock(address beneficiary, uint256 amount, uint256 start, uint256 paymentPlan):** Allows a user to lock funds into a payment plan. This payment plan must not be revoked, and it must be ongoing.
    - Uses *checkIfPaymentPlanHasEnded* and *checkIfPaymentPlanIsRevoked* modifiers.
- **releasableAmount(address beneficiary):** Returns releasable amount for a vestor. This is an external wrapper for the *_releasableAmount* function.
- **release(address beneficiary):** Allows a user to release his/her lock.
    - Uses *checkIfPaymentPlanIsRevoked* modifier.
- **_releasableAmount(Lock storage lock):** Private function that returns a releasable amount for a lock.
- **_vestedAmount(Lock storage lock):** Private function that calculates the vested amount in a given Lock object.
- **revoke(uint256 paymentPlanId):** External function allows a vesting admin to revoke a payment plan.

2.4. Use Cases

1. TokenVesting contract is deployed to a network via a token address. The deployer address is set to be the Vesting admin
2. The vesting admin creates the following payment plans (these plans are all last for 6 months):
    a. 13% allocation (approximately 6,5M tokens) for early investors, which unlocks %8 every month
    b. 4% (approximately 2M tokens) for private sales which unlocks %10 every month
    c. 15% (approximately 7,5M tokens) for the team.
    d. 20% (approximately 10M tokens) for the company reserve.
    e. Rest is for the ecosystem.
3. A vestor interacts with the system and locks a considerable amount of tokens for the Ecosystem plan for 4 months
4. After 4 months the vestor releases his/her funds.