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 lets owners create multiple payment plans and deposit tokens to allow users to receive those funds in a way described by the plan.
The vesting contract allows users to set a start date, end date, and amount of tokens that will be released at the end of the vesting period. This allows users to set a schedule for token releases and ensures that tokens are released over a fixed period of time. Furthermore, the contract also allows users to set a cliff period, which means that no tokens will be released until the cliff period is over. This ensures that users will have a vested interest in the project and will not receive tokens until they have committed to the project for a certain length of time.

1. Functional Requirements

1.1. Roles

Hacken.IO sample project has two roles:

- **Vesting Admin:** All control of the vesting contract belongs to a vesting admin. A vesting admin can create payment plans, and if needed, can revoke a payment plan. Allows locking funds for users.
- **User:** can withdraw funds according to a provided payment plan.

1.2. Features

Hacken.IO Sample Token Vesting has the following features:

- Create new payment plans. (Admin)
- Lock funds (tokens) into payment plans. Each address can have only 1 active vesting plan.
- Revoke payment plans. (Admin)
- Release vested tokens. (User)
- Calculate the amount that should be unlocked. (Everyone)
- Get detailed information about a user's participation in the payment plans. (Everyone)
- Get the number of payment plans. (Everyone)

1.3 Use Cases

1. The vesting admin creates a new vesting plan. A period plan should contain the following information: the duration of 1 vesting period, the total number of periods, and the cliff period.
2. An admin can lock funds for a target user using one of the vesting plans. Tokens for each active user vesting are provided to the vesting contract by the admin through an allowance setting. Since the lock() function is public, any user holding the ERC20 tokens can set-up a lock for other users.
3. Users can withdraw their funds according to a provided payment plan. Tokens can be unlocked after the start of each period in equal parts. If a cliff period is specified, the user should not be able to withdraw funds before it ends. The cliff period accumulates funds to be released, even if they can be released only after it has ended.

2. Technical Requirements

This project has been developed with **Solidity** language, using **Hardhat** as a development environment. **Typescript** is the selected language for testing and scripting.

In addition, **OpenZeppelin**'s libraries are used in the project. All information about the contracts library and how to install it can be found in their GitHub.

In the project folder, the following structure is found:

```
.
├── contracts
│   ├── interfaces
│   │   └── IERC20Detailed.sol
│   ├── mocks
│   │   └── TokenWithVaryingDecimals.sol
│   └── TokenVesting.sol
├── docs
│   └── Requirements.pdf
├── hardhat.config.ts
├── package.json
├── package-lock.json
├── README.md
├── scripts
│   └── deploy.ts
```

```
├── test
│   ├── shared
│   │   ├── accounts.ts
│   │   ├── parser.ts
│   │   └── utilities.ts
│   └── TokenVestingTest.ts
└── tsconfig.json
```

Start with **README.md** to find all basic information about project structure and scripts that are required to test and deploy the contracts.

Inside the **./contracts** folder, **TokenVesting.sol** contains the smart contract with the vesting functionality explained in section 2.2 of this document. **TokenVesting** imports the ERC20 interface **IERC20Detailed**, from the **interfaces** folder that provides the necessary ERC20 methods to run the contract. The additional solidity file **TokenWithVaryingDecimals.sol** is provided in **mocks** in order to run the tests for the project.

In the **./tests** folder, **TokenVestingTest.ts** provides the tests of the different methods of the main contract, in Typescript. In **shared**, different test helpers are found.

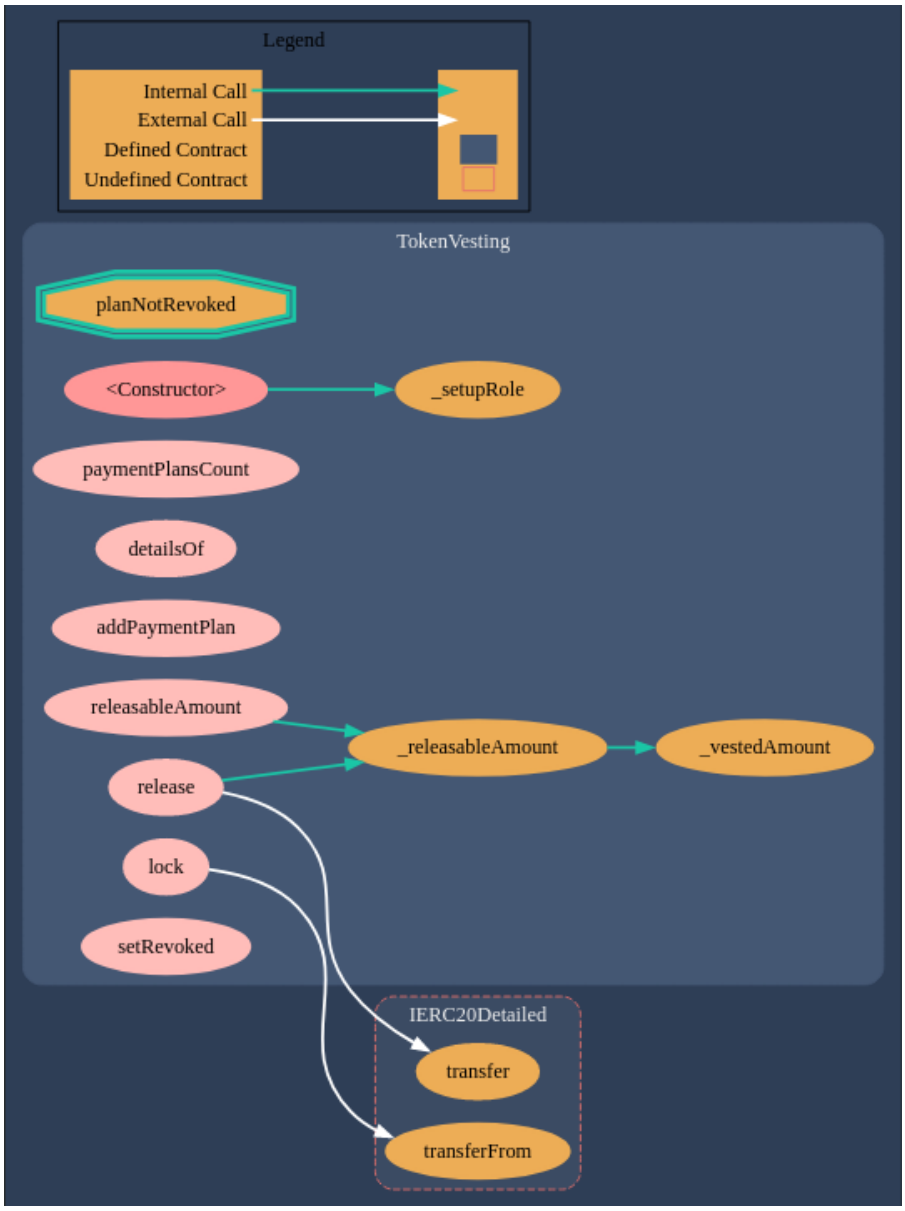
The main contract can be deployed using the **deploy.ts** script in **./deploy**. In order to do so, **.env.example** must be renamed **.env**, and all required data must be provided.

The project configuration is found in **hardhat.config.ts**, where dependencies are indicated. Mind the relationship of this file with **.env**. A basic configuration of Polygon's Mumbai testnet is described in order to deploy the contract. And an etherscan key is set to to configure its different functionalities directly from the repo. More information about this file's configuration can be found in the Hardhat Documentation.

Finally, this document can be found in **./docs**.

2.1. Architecture Overview

The following chart provides a general view of the TokenVesting contract structure and interactions between different functions.



2.2. Contract Information

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

2.2.1. TokenVesting.sol

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

A typical flow looks as follows:

Contract deployment and addition of payment plans

1. The contract is deployed, specifying the ERC20 token that is going to be vested and finally distributed among the different beneficiaries. In the constructor, the deployer's address is set as the admin, following OpenZeppelin's Ownable contract.
2. The admin creates a new payment plan using **addPaymentPlan()**, which will be stored in the paymentPlans array. All the specifications of the payment plan are stored as a PaymentPlan struct.

Lock a beneficiary's vesting

3. Using the **lock()**, a user vesting process can be set up by any account. Note that the contract must be approved to spend the ERC20 vesting token to call this function. This method will create a new **Lock** struct, containing the plan details of the beneficiary, and add it into the **locks** mapping using the address of the user as the key.

Release tokens

4. The vested tokens can be released by calling **release()**. Any user can call this function, which will send out only those tokens that are already unlocked (i.e. the amount released from the accumulated lock periods of the plan). In order to get the amount of tokens that will be released, this method internally calls **_releasableAmount()** and **_vestedAmount()**.

Revoking a plan

5. The admin can revoke any plan by calling the **setRevoked()** method for the target plan, which will be checked to perform any lock.

Getter functions

Additionally, there are different getter functions, that can be called to retrieve relevant data from the contract:

- **paymentPlansCount()** → returns the amount of payment plans.
- **detailsOf()** → returns the Lock and the PaymentPlan of a given beneficiary.

- **releasableAmount()** → returns the tokens that are available to be released for a given beneficiary.

2.2.1.1. Assets

Hacken.IO Sample Token Vesting contains two Structs:

- **PaymentPlan:** This object contains information about a payment plan.
  - uint256 periodLength - length of each period, in seconds.
  - uint256 periods - total vesting periods of the payment plan.
  - uint256 cliff - initial period when no distribution will happen, in seconds. Tokens unlocked during this period will be available for release after the cliff has ended.
  - bool revoked - True if plan has been revoked by admin, false otherwise.
- **Lock:** This structure holds information about a user's participation in a payment plan.
  - address beneficiary - address of the beneficiary to whom vested tokens are transferred.
  - uint256 start - Unix timestamp at which point vesting starts.
  - uint256 paymentPlan - index of the payment plan to apply.
  - uint256 totalAmount - total amount of tokens to be distributed to the beneficiary, in Wei.
  - uint256 released - distributed amount of tokens (so far).

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

- **paymentPlans**: A public PaymentPlan array holding the created payment plans of the Hacken.IO Sample Token Vesting.
- **locks:** mapping that stores the Lock struct of each beneficiary address.
- **token:** ERC20 token address of the vesting token.
- **PERCENT_100:** Constant denominator for %100.
- **VESTING_ADMIN:** keccak256 hash of "VESTING_ADMIN", representing a role of this same name for access control purposes.

2.2.1.2. Events

Hacken.IO Sample Token Vesting has the following events:

- **TokensReleased:** Emitted when tokens are released to beneficiaries. The argument amount represents the amount of released tokens, the argument beneficiary is the address receiving the funds.
- **TokensLocked:** Emitted when tokens are locked. The argument amount represents the amount of locked tokens in Wei, beneficiary is the

address receiving tokens and paymentPlan corresponds to the index of the payment plan in the paymentPlans array.

2.2.1.3. Modifiers

Hacken.IO Sample Token Vesting has the following modifiers:

- **planNotRevoked:** Checks if the given payment plan has been revoked. This modifier first checks that the payment plan index provided is within the payment plans array length, otherwise an out of range error might be thrown.

2.2.1.4. Functions

Hacken.IO Sample Token Vesting has the following functions:

- **constructor:** sets the vesting token address and the VESTING_ADMIN role.
- **paymentPlansCount:** Returns number of payment plans created so far, including revoked plans.
- **detailsOf:** returns both Lock and PaymentPlan data of the input beneficiary address.
- **addPaymentPlan:** adds a new payment plan to paymentPlans array. Only VESTING_ADMIN can call this function.
- **lock:** sets up a vesting process for a beneficiary. Allows a user to lock funds into a payment plan. This payment plan must not be revoked.
- **releasableAmount:** returns the amount of tokens that can be unlocked for a given user.
- **release:** releases the lock of a user, if available, updating the lock's released amount and transferring the tokens to the beneficiary.
- **setRevoked:** revokes a payment plan. Only VESTING_ADMIN can call this function.
- **_releseableAmount:** private method that returns the releasable amount of tokens for a given lock.
- **_vestedAmount:** private method that returns the vested tokens according to the lock periods that have passed.