Wormfare WOFR Smart Contract

Project Overview

The WOFR contract is an ERC20 token smart contract with a fixed supply of 300,000,000 tokens. The token name is “Wormfare” and the ticker is “WOFR”. The entire supply is pre-minted to a specified wallet address upon deployment. The contract is based on OpenZeppelin’s ERC20 implementation and includes the following extensions: ERC20Permit and ERC20Burnable.

1. Functional Requirements

1.1 Roles

1. **Owner**: The initial token holder, who holds the entire token supply upon deployment. The owner is responsible for transferring tokens to other accounts as needed.
2. **Token Holder**: Any account holding WOFR tokens. Token holders can transfer tokens, permit allowances, or burn their own tokens.

1.2 Features

The WOFR contract includes the following features:

- **ERC20 Standard Token Functions**: Provides standard ERC20 token functionality, including transfer, approve, and allowance methods.
- **ERC20Permit**: Allows token holders to set allowances via signatures instead of direct transactions, which is gas-efficient and enables use with off-chain platforms.
- **ERC20Burnable**: Enables any token holder to burn (destroy) their own tokens, reducing the total supply.
- **Pre-Minted Supply**: Upon deployment, the total supply is assigned to a specified wallet address.

2. Technical Requirements

This project is developed using Solidity, with Hardhat as the development environment and TypeScript for testing and deployment scripting. OpenZeppelin's libraries are used for the ERC20 base implementation and extensions.

In the project folder, the following structure is found:

```
├── audits
│   └── TokenSale.pdf
├── contracts
│   ├── nft
│   │   └── WormfareGenesis.sol
│   ├── spinner
│   │   └── Spinner.sol
│   ├── testing
│   │   ├── Tether.sol
│   │   └── WOFRTest.sol
│   ├── TokenSale.sol
│   └── WOFR.sol
├── deploy
│   └── contracts
│       ├── nft
│       │   └── 01_WormfareGenesis.ts
│       ├── testing
│       │   ├── 01_Tether.ts
│       │   └── 02_WOFRTest.ts
│       ├── utils
│       │   └── deployment-utils.ts
│       ├── 01_TokenSale.ts
│       ├── 02_TokenSalePrivate.ts
│       ├── 03_TokenSaleKol.ts
│       ├── 04_Spinner.ts
│       └── 05_WOFR.ts
├── docs
│   ├── api
│   │   ├── nft
│   │   │   └── WormfareGenesis.md
│   │   ├── testing
│   │   │   └── Tether.md
│   │   └── TokenSale.md
│   ├── assets
│   │   ├── TokenSale_case1.png
│   │   ├── TokenSale_case2.png
│   │   └── TokenSale_case3.png
│   ├── Spinner.md
│   ├── TokenSale.md
│   ├── WOFR.md
│   ├── Wormfare TokenSale Smart Contract.pdf
│   ├── Wormfare WOFR Smart Contract.pdf
│   └── WormfareGenesis.md
├── src
│   ├── enums
│   │   └── network.enum.ts
│   ├── zod
│   │   ├── zod-helpers.ts
│   │   └── zod-rules.ts
│   └── utils.ts
├── templates
│   ├── contract.hbs
│   ├── event.hbs
│   ├── function.hbs
│   ├── helpers.js
│   └── page.hbs
├── test
│   ├── deploy
│   │   ├── spinner.deploy.ts
│   │   ├── tether.deploy.ts
│   │   ├── token-sale.deploy.ts
│   │   ├── wofr.deploy.ts
│   │   └── wormfare-genesis.deploy.ts
│   ├── utils
│   │   ├── common.ts
│   │   └── constants.ts
│   ├── spinner.test.ts
│   ├── token-sale.test.ts
│   ├── wofr.test.ts
│   └── wormfare-genesis.test.ts
├── Dockerfile
├── README.md
├── hardhat.config.ts
├── hardhat.validate.ts
├── package.json
├── tsconfig.json
└── yarn.lock
```

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

Inside the **./contracts** folder, **WOFR.sol** contains the smart contract that this document describes.

In the **./test** folder, **wofr.test.ts** provides the tests of the different methods of the main contract, in Typescript. In the **./test/deploy** folder, **wofr.deploy.ts** provides the contract deployment logic used in the tests.

The contract can be deployed using the **05_WOFR.ts** script in the **./deploy/contracts** folder. In order to do so, **.env.example** must be renamed to **.env**, and all required data must be provided.

The project configuration is found in **hardhat.config.ts**, where dependencies are indicated. **hardhat.validate.ts** provides the **.env** validation rules. Mind the relationship of those files with **.env.example**. A basic configuration for the Hardhat test network deployment is provided. More information about this file's configuration can be found in the Hardhat Documentation.

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

2.2. Contract Information

2.2.1. WOFR.sol

The WOFR contract provides basic ERC20 functionality with additional support for permit-based allowances and token burning.

A typical flow looks as follows:

Contract deployment

The contract is deployed, specifying the owner's wallet address that is going to hold the whole supply of tokens.

Token distribution

The owner distributes the tokens to other wallets and smart contracts.

2.2.1.1. Assets

Some of the post-deployment contract parameters look as follows:

- **totalSupply**: The fixed total supply of 300,000,000 tokens.
- **name**: "Wormfare"
- **symbol**: "WOFR"

2.2.1.2. Events and Modifiers

All the events, modifiers, and other stuff can be found in the OpenZeppelin's ERC20 token API spec: https://docs.openzeppelin.com/contracts/5.x/api/token/erc20

2.2.1.3. Functions

All the functions can be found in the OpenZeppelin's ERC20 token API spec: https://docs.openzeppelin.com/contracts/5.x/api/token/erc20.
Only the constructor will be described here:

- **constructor:** Deploys the contract, initializing the total supply, token name, and symbol. Mints the total supply to a specified wallet address.