Bull and Bear Exchange

DMBLOCK Assignment 2

Lukáš Častven, Jakub Jelinek
Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií
xcastven@stuba.sk, xjelinek@stuba.sk

April 27, 2024

Contents

1 Assignment **2**

2 Questions **2**

3 Implementation **2**

4 Testing **3**

5 Security analysis **3**

6 Conclusion **3**

1 Assignment

Main goal of this assignment was to complete provided implementation of an Uniswap [1] inspired decentralized exchange. We were given solidity and javascript source codes in a Hardhat project, and we had to implement methods in these files to create a functional decentralized exchange for swapping Ether with our custom ERC20 token.

2 Questions

1. Why removing liquidity from exchange doesn't change the rate

2. Explain implemented fee mechanism for incentivizing liquidity providers

3. Explain at least one gas optimisation method you used

Feedback questions

4. How much time did you spend on the assignment

5. What would one useful information before you started to work on this assignment

6. What would one thing you would change

3 Implementation

We decided to rewrite the provided Hardhat project into Foundry [2]. Our decentralized exchange is called **Bull & Bear Exchange** and the ERC20 token traded on this exchange is **Bull & Bear Token**. Also we rewrote provided web app into Vue [3].

The project structure looks like this:

- `app` - contains the Vue frontend interacting with the smart contracts
- `dex` - contains the Foundry project for the smart contracts of **Bull & Bear Exchange**
- `docs` - contains documentation for this assignment

Smart contracts

Bull & Bear Token — BBT

ERC20 token to be traded on our exchange is called **Bull & Bear token**, with symbol being **BBT**. We argue that the two functions from assignment (`mint` and `disable_mint`), which we have to implement, are useless and potentially an anti-pattern. ERC20 implementation by OpenZeppelin is enough to implement a token with constant supply. By pre-minting supply to the deployer of the token, we achieved a token with constant supply (no new tokens can be minted as `_mint` in ERC20

is an internal function [4]). Thus we have reduced the complexity of this token implementation by removing `mint`, `disable_mint` and even the `Ownable` parent contract used in provided source code.

Thus the whole contract has few lines and minimal complexity:

```
import {ERC20} from "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract BBToken is ERC20 {
    constructor(uint256 supply) ERC20("Bull and Bear Token", "BBT") {
        _mint(msg.sender, supply * 10 ** decimals());
    }

    function decimals() public pure override returns (uint8) {
        return 0;
    }
}
```

The assignment requires that our token be indivisible, the function decimals is overridden to reflect this requirement.

4 Testing

5 Security analysis

6 Conclusion

References

[1] "Overview — Uniswap — docs.uniswap.org." https://docs.uniswap.org/contracts/v3/overview. [Accessed 27-04-2024].

[2] "Foundry Book — book.getfoundry.sh." https://book.getfoundry.sh/. [Accessed 27-04-2024].

[3] "Vue.js — vuejs.org." https://vuejs.org/. [Accessed 27-04-2024].

[4] "ERC 20 - OpenZeppelin Docs — docs.openzeppelin.com." https://docs.openzeppelin.com/contracts/4.x/api/token/erc20#ERC20-_mint-address-uint256-. [Accessed 27-04-2024].