DEFI Crash Course: DEX Exercise 2 - Sniper

Intro

You got a call from your rich friend who researched MEV (Miner-Extractable Value)!

He asked you to help him setup some Ethereum Validator nodes and create a liquidity "Sniper Bot".

When a DEX pair is created and liquidity is added, this Sniper Bot will attempt to buy it as soon as possible.

During this exercise, you'll develop a liquidity sniper bot called "Sniper".

You'll need to work directly with the low-level UniswapV2 pair smart contract to be quick and precise.

A new "Precious Token" is launching on MAINNET, which is perfect for testing your bot!

According to the Precious Token team, they will add liquidity of 10,000 PRECIOUS tokens and 50 WETH.

The initial price is 50 / 10,000 = 0.005 ETH per `PRECIOUS`.

Your first goal is to make sure you can purchase the tokens before anyone else

and the second goal is to purchase the tokens at the cheapest price possible.

Note: This exercise is executed on an Ethereum mainnet Fork block number `15969633`. Everything is already configured in the `hardhat.config.js` file

Ethereum MAINNET Addresses

```
Uniswap V2 Factory: 0x5c69bee701ef814a2b6a3edd4b1652cb9cc5aa6f
WETH Token: 0xc02aaa39b223fe8d0a0e5c4f27ead9083c756cc2

Your Rich Friend's Account (To be impersonated):
0x8e5dedeaeb2ec54d0508973a0fccd1754586974a
```

Accounts

- 0 - Liquidity Adder
- 1 - You

Tasks

Task 1 - Contract Development

Complete all the open TODOs in the `./contracts/dex-2/Sniper.sol` smart contract.

Implement all the functions:

External `snipe()` function:

- The external function which initiates the snipe.
- Receives `_tokenIn`, `_tokenOut`, `_amountIn`, `_absoluteMinAmountOut`, `_maxRetries`.
- Fetches the pair address from the Uniswap Factory (reverts if pair wasn't created yet).
- Sort the tokens using the internal `_sortTokens` function.
- Get the pair reserves.
- Get the exepcted amount out the we will receive using the internal `_getAmountOut`, reverts if it's lower then `_absoluteMinAmountOut`.
- Sends the `_amountIn` in `_tokenIn` token to the pair smart contract.
- Execute the swap and retries `_maxRetries` times, every retry with higher slippage tolerance (0.3% increments).

Internal `_sortTokens` function:

- Sort two tokens based on their address.
- You should implement the exact same logic like that exists in the Unsiwap Factory `createPair()` function.

Internal `_getAmountOut` function:

- Returns the expected amount of tokens which we will receive based on given amountIn and pair reserves.
- You should implement the exact same logic like that exists in the Unsiwap Library `_getAmountOut()` function.

Internal `_swapWithRetries` function:

- Swaps the tokens until it succeded or reached `_maxRetries` (Call itself in recursio).
- Starts with 0% slippage tolerance (`INITIAL_SLIPPAGE`),
- Increases the slippage tolerance every failed attemps by 0.3% (`SLLIPAGE_INCREMENTS`).
- Calls the low-level `swap()` function in the pair smart contract to swap the tokens

Task 2 - Tests

Complete all the open TODOs in `test/dex-2/tests.js`

Deployment

- Deploy the `Sniper.sol` smart contract

Snipe

- User your smart contract to snipe `Precious` tokens, invest 35 ETH.
- The maximum price that you're willing to pay for a token is 0.02 ETH (4 times greater than the initial price)
- Call you snipe function with 3 retries as parameter.